UNCLASSIFIED

AD NUMBER

ADB222369

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution: Further dissemination only as directed by U.S. Army Strategic Defense Command, Attn: SCCD-IM-PA, P.O. Box 1500, Huntsville, AL 35807-3801, Jan 97 or higher DoD authority.

AUTHORITY

Phillips Lab. [AFMC] Kirtland AFB, NM ltr dtd 30 Jul 97

THIS PAGE IS UNCLASSIFIED

PL-TR-96-1126, Pt. 1

PL-TR-
96-1126,
Pt 1

ADVANCED HARD REAL-TIME OPERATING SYSTEM, THE MARUTI PROJECT

Part 1 of 2

Ashok K. Agrawala
Satish K. Tripathi

Department of Computer Science
University of Maryland
College Park, MD 20742

January 1997

Final Report

Further dissemination only as directed by the U.S. Army Strategic Defense Command, ATTN: SCCD-IM-PA, P.O. Box 1500, Huntsville, AL 35807-3801, January 1997, or higher DoD authority.

WARNING - This document contains technical data whose export is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec 2751 et seq.) or The Export Administration Act of 1979, as amended (Title 50, U.S.C., App. 2401, et seq.). Violations of these export laws are subject to severe criminal penalties. Disseminate IAW the provisions of DoD Directive 5230.25 and AFI 61-204.

DESTRUCTION NOTICE - For classified documents, follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

PHILLIPS LABORATORY
Space Technology Directorate
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

PL-TR-96-1126

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

If you change your address, wish to be removed from this mailing list, or your organization no longer employs the addressee, please notify PL/VTS, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776.

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

This report has been approved for publication.

James E Russell

JIM RUSSELL, Capt, USAF
Project Manager

FOR THE COMMANDER

Nancy Crowley

NANCY L. CROWLEY, Lt Col, USAF
Chief, Satellite Control and Simulation Division

CHRISTINE M. ANDERSON, SES
Director, Space Technology

DRAFT SF 298

1. Report Date (dd-mm-yy)
January 1997

2. Report Type
Final

3. Dates covered (from... to)
4/92 to 10/96

4. Title & subtitle
Advanced Hard Real-Time Operating System, The Maruti Project

5a. Contract or Grant #
DASG-60-92-C-0055

5b. Program Element # 62301E

6. Author(s)
Ashok K. Agrawala
Satish K. Tripathi

5c. Project # DRPB

5d. Task # TB

5e. Work Unit # AT

7. Performing Organization Name & Address
Department of Computer Science
University of Maryland
College Park, MD 20742

8. Performing Organization Report #

9. Sponsoring/Monitoring Agency Name & Address
Phillips Laboratory
3550 Aberdeen Ave. SE
Kirtland, AFB, NM 87117-5776

10. Monitor Acronym

11. Monitor Report #
PL-TR-96-1126, Part 1

12. Distribution/Availability Statement
Further dissemination only as directed by the U.S. Army Strategic Defense Command, ATTN: SCCD-IM-PA, P.O. Box 1500, Huntsville, AL 35807-3801, January 1997, or higher DoD authority.

13. Supplementary Notes

14. Abstract System correctness for real-time systems relies on both functional and temporal correctness of the system components. In order to allow creation and deployment of critical applications with hard real-time constraints in a reactive environment, we have developed the Maruti environment, which consists of the Maruti operating system and runtime environment, and an application development and environment that uses the Maruti Programming Language (MPL), an extension of ANSI C; the Maruti Configuration language (MCL), which specifies how MPL modules are to be connected and any environmental constraints; and various analysis and debugging tools. The core of the Maruti runtime system is the Elemental Unit (EU) and calendar. An EU is an atomic entity triggered by incoming data/signals, that produces data/signals. A calendar specifies the execution order and time for each EU. Calendars are static entities created during application design and development, thus allowing temporal debugging of applications before they are executed on the machine. A given application may have more than one calendar to allow contingency or degraded operation.

15. Subject Terms
Real-Time operating systems, fault tolerance, concurrency, embedded systems, environments

Security Classification of

16. Report
Unclassified

17. Abstract
Unclassified

18. This Page
Unclassified

19. Limitation of Abstract
Limited

20. # of Pages
220

21. Responsible Person (Name and Telephone #)
Capt Jim Russell
(505) 846-8986 ext 352

TABLE OF CONTENTS

Executive Summary

Introduction:

This is the final report on the work done under contract DASG-60-92-C-0055 from Phillips Labs and ARPA to the Department of Computer Science at the University of Maryland. The work started 04/28/92. The goal of this project was to create an environment for development and deployment of critical applications with hard real-time constraints in a reactive environment . We have redesigned Maruti system to address these issues. In this report we highlight the achievements of this contract. A publications list and a copy of each of the publications is also attached.

Application Development Environment:

To support applications in a real-time system, conventional application development techniques and tools must be augmented with support for specification and extraction of resource requirements and timing constraints, The application development system provides a set of programming tools to support and facilitate the development of real-time applications with diverse requirements. The Maruti Programming Language (MPL) is used to develop induvidual program modules. The Maruti Configuration Language (MCL) is used to specify how individual program modules are to be connected together to form an application and the details of the hardware of which the application is to be executed.

In the current version, the base programming language used is ANSI C. MPL adds *modules*, *shared memory blocks*, *critical regions*, *typed message passing*, *periodic functions*, and *message-invoked functions* to the C language. To make analyzing the resource usage of programs feasible, certain C idioms are not allowed in MPL; in particular, recursive function calls are not allowed nor are unbounded loops containing externally visible events, such as message passing and critical region transition.

MPL Modules are brought together into as an executable application by a specification file written in the Maruti Configuration Language (MCL). The MCL specification determines the application's hard real-time constraints, the allocation of tasks, threads, and shared memory blocks, and all message-passing connections. MCL is an interpreted C-like language rather than a declarative language, allowing the instantiation of complicated subsystems using loops and subroutines in the specification.

Analysis and Resource Allocations:

The basic building block of the Maruti computation model is the elemental unit (EU). In general an elemental unit is an executable entity which is triggered by incoming data and signals, operates on the input data, and produces some output data and signals. The behavior of an EU is atomic with respect to its environment. Specifically:

- All resources needed by an elemental unit are assumed to be required for the entire length of its execution.
- The interaction of an EU with other entities of the system occurs either before it starts executing or after it finishes execution.

In order to define complex executions , the EUs may be composed together and properties specified on the composition. Elemental units are composed by connecting an output port of an EU with an input port of another EU. A valid connection requires that the input and output of port types are compatible, i.e., they carry the same message type. Such a connection marks a one-way flow of data or control, depending on the nature of the ports. A composition of EUs can be viewed as a directed acyclic graph, called an *elemental unit graph* (EUG), in which the nodes are the EUs, and the edges are the connections between EUs. An incompletely specified EUG in which all input and output ports are not connected is termed as a *partial* EUG (PEUG). A partial EUG may be viewed as a higher level EU. In a complete EUG, all input and output ports are connected and there are no cycles in the graph. The acyclic requirements come from the required time determinacy of execution. A program with unbounded cycles or recursions may not have a temporally determinate execution time. Bounded cycles in an EUG are converted into a acyclic graph by loop unrolling.

Program modules are independently compiled. In addition to the generation of the object code, compilation also results in the creation of partial EUGs for the modules, i.e., for the services and entries in the module, as well as the extraction of resource requirements such as stack sizes or threads, memory requirements, and the logical resource requirements.

Given an application specification in the Maruti Configuration Language and the component application modules, the integration tools are responsible for creating a complete application program and extracting out the resource and timing information for scheduling and resource allocation. The input of the integration process are the program modules, the partial EUGs corresponding to the modules, the application configuration specification, and the hardware specifications. The outputs of the integration process are: a specification for the loader for creating tasks, populating their address space, creating the threads and channels, and initializing the task; loadable executables of the program; and the complete application EUG along with the resource description for the resource allocation and the scheduling subsystem.

After the application program has been analyzed and its resource requirements and execution constraints identified, it can be allocated and scheduled for a runtime system.

We consider the static allocation and scheduling in which a task is the finest granularity object of allocation and an EU instance is the unit of scheduling. In order to make the execution of instances satisfy the specification and meet the timing constraints, we consider a scheduling frame whose length is the least common multiple of all tasks' periods. As long as one instance of each EU is scheduled in each period within the scheduling frame and these executions meet the timing constraints, a feasible schedule is obtained.

Maruti Runtime System:

The runtime system provides the conventional functionality of an operating system in a manner that supports the timely dispatching of jobs. There are two major components of the runtime system - the *Maruti core*, which is the operating system code that implements scheduling, message passing, process control, thread control, and low level hardware control, and the *runtime dispatcher*, which performs resource allocation and scheduling or dynamic arrivals.

The core of the Maruti hard real-time runtime system consists of three data structures:

- The *calendars* are created and loaded by the dispatcher. Kernel memory is reserved for each calendar at the time it is created. Several system calls serve to create, delete, modify, activate, and deactivate calendars.

- The *results table* holds timing and status results for the execution of each elemental unit; The maruti_calandar_results system call reports these results back up to the user level, usually the dispatcher. The dispatcher can then keep statistics or write a trace file.

- The *pending activation table* holds all outstanding calendar activation and deactivation requests. Since the requests can come from before the switch time, the kernel must track the requests and execute them at the correct time in the correct order.

The Maruti design includes the concept of scenarios, implemented at runtime as sets of alternative calendars that can be switched quickly to handle an emergency or a change in operating mode. These calendars are pre-scheduled and able to begin execution without having to invoke any user-level machinery. The dispatcher loads the initial scenarios specified by the application and activates one of them to begin normal execution.

Optimal Replication of Series-Parallel Graphs for Computation-Intensive Applications*

Sheng-Tzong Cheng
Ashok K. Agrawala
Institute for Advanced Computer Studies
Systems Design and Analysis Group
Department of Computer Science
University of Maryland, College Park, MD 20742
{stcheng,agrawala}@cs.umd.edu

*This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Honeywell or Army/Phillips.

Optimal Replication of SP Graphs for Computation-Intensive Applications

Prof. Ashok K. Agrawala
Department of Computer Science
University of Maryland at College Park
A.V. Williams Building
College Park, Maryland 20742
(Tel): (301) 405-2525

Abstract

We consider the replication problem of series-parallel (SP) task graphs where each task may run on more than one processor. The objective of the problem is to minimize the total cost of task execution and interprocessor communication. We call it, the ***minimum cost replication problem for SP graphs*** (MCRP-SP). In this paper, we adopt a new communication model where the purpose of replication is to reduce the total cost. The class of applications we consider is computation-intensive applications in which the execution cost of a task is greater than its communication cost. The complexity of MCRP-SP for such applications is proved to be NP-complete. We present a branch-and-bound method to find an optimal solution as well as an approximation approach for suboptimal solution. The numerical results show that such replication may lead to a lower cost than the optimal assignment problem (in which each task is assigned to only one processor) does. The proposed optimal solution has the complexity of $O(n^2 2^n M)$, while the approximation solution has $O(n^4 M^2)$, where n is the number of processors in the system and M is the number of tasks in the graph.

1 Introduction

Distributed computer systems have often resulted in improved reliability, flexibility, throughput, fault tolerance and resource sharing. In order to use the processors available in a distributed system, the tasks have to be allocated to the processors. The allocation problem is one of the basic problems of distributed computing whose solution has a far reaching impact on the usability and efficiency of a distributed system. Clearly, the tasks of an application have to be executed satisfying the precedence and other synchronization constraints among them. (Such constraints are often specified in the form of a task graph.)

In executing an application, defined by its task graph, we have the option of restricting ourselves to having only one copy of each task. The allocation problem, in this case, is referred to as *assignment problem*. If, on the other hand, a task may be replicated multiple times, the general problem is called the *replication problem*. In this paper, we consider the replication problem and present an algorithm to find the optimal replication of series-parallel graphs for computation-intensive applications.

For distributed processing applications, the objective of the allocation problem may be the minimum completion time, processor load balancing, or total cost of execution and communication, etc. For the assignment problem where the objective is to minimize the total cost of execution and interprocessor communication, Stone [11] and Towsley [12] presented $O(n^3M)$ algorithms for tree-structure and series-parallel graphs, respectively, of M tasks and n processors. For general task graphs, the assignment problem has been proven [9] to be NP-complete. Many papers [8][9][10] presented branch-and-bound methods which yielded an optimal result. Other heuristic methods have been considered by Lo [7] and Price and Krishnaprasad [5]. All these works focused on the assignment problem.

Traditionally, the main purpose of replicating a task on multiple processors is to increase the degree of fault tolerance [2][6]. If some processors in the distributed system fail, the application may still survive using other copies. In such a communication model, a task has to communicate with multiple copies of other tasks. As a consequence, the total cost of execution and communication of the replication problem will be bigger than that of the assignment problem. In this paper, we adopt another communication model in which the replication of a task is not for the sake of fault tolerance but for decreasing of the total cost. In our model, each task may have more than one copy and it may start its execution after receiving necessary data from one copy of each preceding task. Clearly, in a heterogeneous environment the cost of execution of a task depends on the processor on which it executes, and the communication costs depend on the topology, communication medium, protocols used, etc. When a task t is allowed to have only one copy in the system, the sum of the interprocessor communication costs between t and other tasks may be large. Sometimes it will be more beneficial if we replicate t onto multiple processors to reduce the inter-processor communication, and to fully utilize the available processors in the systems. Such replication may lead to a lower total cost than the optimal assignment problem does. An example illustrating this point is presented in Section 3.

In the assignment problem, polynomial-time algorithms exist for special cases, such as tree-structure [11] and series-parallel [12] task graphs. This paper represents one of the first few attempts at finding special cases for the replication problem. The class of applications we consider in this paper is computation-intensive applications in which the execution cost of a task is greater than its communication cost. Such applications can be found in an enormous number of fields, such as digital signal processing, weather forecasting, game searching, etc. We formally define a computation-intensive application in Section 2.2. In this paper, we prove that for the computation-intensive applications, the replication problem is NP-complete, and we present a branch-and-bound algorithm to solve it. The worst-case complexity of the solution is $O(n^2 2^n M)$. Note that the algorithm is able to solve the problem in the complexity of the linear function of M.

We also develop an approximation approach to solve the problem in polynomial time. Given a forker task s with K successors in the SP graph, the method tries to allocate s to processors based on iterative selection. The complexity of the iterative selection for a forker is $O(n^2 K^2)$, while the overall solution for an SP graph is $O(n^4 M^2)$.

In the remainder of this paper, the series-parallel graph model and the computation model are described in section 2. In section 3, the replication problem is formulated as the minimum cost 0-1 integer programming problem and the proof of NP completeness is given. A branch-and-bound algorithm and numerical results are given in section 4, while the approximation methods and results are given in section 5. The overall algorithm is presented and conclusion remark is drawn in section 6.

2 Definitions

2.1 Graph Model

A *series-parallel* (SP) graph, $G = (V, E)$, is a directed graph of type p, where $p \in \{T_{unit}, T_{chain}, T_{and}, T_{or}\}$ and G has a source node (of indegree 0) and a sink node (of outdegree 0). An SP graph can be constructed by applying the following rules recursively.

1. A graph $G = (V, E) = (\{v\}, \phi)$ is an SP graph of type T_{unit}. (Node v is the source and the sink of G.)

2. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are SP graphs then $G' = (V', E')$ is an SP graph of type T_{chain}, where $V' = V_1 \cup V_2$ and $E' = E_1 \cup E_2 \cup \{<$sink of G_1, source of $G_2 >\}$.

3. If each graph $G_i = (V_i, E_i)$ with source-sink pair (s_i, t_i), where s_i is of outdegree 1, is an SP graph, $\forall\, i = 1, 2, \ldots, n$, and new nodes $s' \notin V_i$ and $t' \notin V_i$, $\forall\, i$ are given then $G' = (V', E')$ is an SP graph of type T_{and}(or type T_{or}), where $V' = V_1 \cup V_2 \cup \ldots \cup V_n \cup \{s', t'\}$ and $E' = E_1 \cup E_2 \cup \ldots \cup E_n \cup \{< s', s_i > \mid \forall\, i = 1, 2, \ldots, n\} \cup \{< t_i, t' > \mid \forall\, i = 1, 2, \ldots, n\}$. The source of G', s', is called the *forker* of G'. The sink of G', t', is called the *joiner* of G'. G' is an SP graph of type T_{and}(or type T_{or}) if there exists a *parallel-and* (or *parallel-or*) relation among G_i's.

A convenient way of representing the structure of an SP graph is via a parsing tree [4]. The transformation of an SP graph to a parsing tree can be done in a recursive way. There are four kinds of internal nodes in a parsing tree: T_{unit}, T_{chain}, T_{and} and T_{or} nodes. A T_{unit} node has only one child, while a T_{chain} node has more than one child. Every internal node x, along with all its descendant nodes induces a subtree S_x which describes an SP subgraph G_x of G. Each leaf node in S_x corresponds to an SP graph of type T_{unit}. A T_{and}(or T_{or}) node y consists of its type T_{and}(or T_{or}) along with the forker and joiner nodes of G_y. We give an example of an SP graph G, and its parsing tree $T(G)$ in Figure 1.

2.2 Computational Model

An application program consists of M tasks labeled $m = 1, 2, \ldots, M$. Its behavior is represented by an SP graph with the tasks correspond to the nodes. Each task may be replicated onto more than one processor. A *task instance* $t_{i,p}$ is a replication of task i on processor p. A directed edge $< i, j >$ between nodes i and j exists if the execution of task j follows that of task i. Associated with each edge $< i, j >$ is the communication cost incurred by the application. We are concerned with types of applications where the cost of execution of a task is always greater than the communication overhead it needs. The model is stated as follows.

Given a distributed system S with n processors connected by a communication network, an application is *computation-intensive* if its associated SP graph $G = (V, E)$ on S satisfies the following conditions:

1. $\mu_{i,j}(p, q) \geq 0$,

2. $\sum_{q=1}^{n} \mu_{i,j}(p, q) \leq \min_p(e_{i,p})$, $\forall < i, j > \in E$, and $1 \leq p \leq n$, where

- $\mu_{i,j}(p, q)$ is the communication cost between tasks i and j when they are assigned to processors p and q respectively, and

- $e_{i,p}$ is the execution cost when task i is assigned to processor p.

The first condition states that the communication cost between any two task instances (e.g. $t_{i,p}$ and $t_{j,q}$) is not negative. The second one depicts that for every edge $< i, j >$, the worst-case communication cost between any task instance $t_{i,p}$ and all its successor task instances (i.e. $t_{j,q}$'s, $\forall$ q) is less than the minimum execution cost of task i.

2.3 Communication Model

The communication model we considered is different from that of reliability-oriented replication. In reliability-oriented replication problem, the objective is to increase the degree of fault tolerance. To detect fault and maintain data consistency, each task has to receive multiple copies of data from several task instances if its predecessor is replicated in more than one place.

The purpose of the replication problem considered in this paper is to decrease the sum of execution and communication costs. Under such consideration, there is no need to enforce plural communication between any two task instances. Hence, we propose the *1-out-of-n* communication model. In the model, for each edge $< i, j > \in E$, a task instance $t_{j,q}$ may start its execution if it receives the data from *any one* task instance of its predecessor, task i.

3 Problem Formulation and Complexity

Based on the computational model presented in Section 2.2, the problem of minimizing the total sum of execution and communication costs for an SP task graph can be approached by replication of tasks. An example where the replication may lead to a lower sum of execution costs and communication costs is given in Figure 2, where the number of processors in the system is two, and the execution costs and communication costs are listed in e table and μ table respectively. If each task is allowed to run on at most one processor, then the optimal allocation will be to assign task a to processor 1, b to 1, c to 1, d to 2, e to 2, and f to 1. The minimum cost is 68. However, if each task is allowed to be replicated more than one copies, (i.e. to replicate task a to processors 1 and 2), then the cost is 67.

We introduce integer variable $X_{i,p}$'s, $\forall\ 1 \leq i \leq M$ and $1 \leq p \leq n$, to formulate the problem where each $X_{i,p} = 1$ if task i is replicated on processor p; and $= 0$, otherwise. We define a binary function $\delta(x)$. If $x > 0$ then $\delta(x) = 1$ else $\delta(x) = 0$. We also associate an *allocated flag* $F(w)$ with each node w in the parsing tree, where $F(w) = 1$ if the allocation for tasks in the subtree S_w is valid; and $= 0$, otherwise. A valid allocation for the tasks in S_w is an allocation that follows the semantics of T_{chain}, T_{and}, and T_{or} subgraphs. A valid allocation is not necessarily the allocation in which each task in S_w is allocated to at least one processor. Some tasks in T_{or} subgraphs may be neglected without effecting the successful execution of an SP graph.

Given an SP graph G, its parsing tree $T(G)$ and any internal node w in $T(G)$, allocated flag $F(w)$ can be recursively computed:

1. if w is a T_{unit} node with a child i, then

$$F(w) = F(i) = \delta(\sum_{p=1}^{n} X_{i,p})$$

2. if w is a T_{chain} node with c children, $F(w) = F(child_1) \times F(child_2) \times \ldots \times F(child_c)$.

3. if w is a T_{and} node with forker s, joiner t and c children, then $F(w) = F(s) \times F(t) \times F(child_1) \times F(child_2) \times \ldots \times F(child_c)$.

4. if w is a T_{or} node with forker s, joiner t and c children, then $F(w) = F(s) \times F(t) \times \delta(F(child_1) + F(child_2) + \ldots + F(child_c))$.

The minimum cost replication problem for SP graphs, MCRP-SP, can be formulated as 0-1 integer programming problem, i.e:

$$Z = \text{Minimize } [\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E,\ 1\leq q\leq n} \min_{X_{i,p}=1} (\mu_{i,j}(p,q) * X_{j,q}) \ ]$$

$$\text{subject to } F(\tau) = 1, \text{ where } \tau \text{ is the root of } T(G) \text{ and } X_{i,p} = 0 \text{ or } 1, \forall i,p. \qquad (1)$$

The restricted problem which allows each task to run on at most one processor has the following formulation.

$$Z = \text{Minimize } [\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j>\in E, p,q} \mu_{i,j} * X_{i,p} * X_{j,q} \ ]$$

$$\text{subject to } \sum_{p=1}^{n} X_{i,p} \leq 1 \text{ and } F(\tau) = 1,$$

$$\text{where } \tau \text{ is the root of } T(G) \text{ and } X_{i,p} = 0 \text{ or } 1, \forall i,p. \qquad (2)$$

The task assignment problem (2) for SP graphs of M tasks onto n processors, has been solved in $O(n^3M)$ time [12]. However,the multiprocessor task assignment for general types of task graphs without replication has been reported to be NP-complete [9]. As for the MCRP-SP problem, it can be shown to be NP-complete. In this paper, we are able to solve the problem and present a linear-time algorithm that is linear in the number of tasks when the number of processors is fixed for computation-intensive applications.

3.1 Assignment Graph

Bokhari [1] introduced the assignment graph to solve the task assignment problem (2). To prove the NP completeness of problem (1) and solve the problem, we also adopt the concept of the assignment graph of an SP graph. The assignment graph of an SP graph can be defined similarly. The following definitions apply to the assignment graph. And we draw up an assignment graph for an SP graph in Figure 3.

1. It is a directed graph with weighted nodes and edges.
2. It has $M \times n$ nodes. Each weighted node is labeled with a task instance, $t_{i,p}$.
3. A *layer* i is the collection of n weighted nodes ($t_{i,1}$, $t_{i,2}$, ..., and $t_{i,n}$). Each layer of the graph corresponds to a node in the SP graph. The layer corresponding to the source (sink) is called source (sink) layer.
4. A part of the assignment graph corresponds to an SP subgraph of type T_{chain}, T_{and} or T_{or} is called a T_{chain}, T_{and} or T_{or} *limb* respectively.
5. Communication costs are accounted for by giving the weight $\mu_{i,j}(p,q)$ to the edge going from $t_{i,p}$ to $t_{j,q}$.
6. Execution costs are assigned to the corresponding weighted nodes.

Given an assignment graph, Bokhari [1] solves Problem (2) by selecting one weighted node from each layer and including the weighted edges between any two selected nodes. This resulting subgraph is called an *allocation graph*. To solve Problem (1), more than one weighted node from each layer may be chosen. Similarly, a *replication graph* for Problem (1) can be constructed from an assignment graph by including all selected nodes and edges between these nodes. Examples of an allocation graph and a replication graph are shown in Figure 4 for an assignment graph shown in Figure 3. Note that for each node x in the replication graph there is only one edge incident to it from each predecessor layer of x.

In a replication graph, each layer may have more than one selected node. Let Variable $\bar{X}_l$ = ($X_{l,1}$, $X_{l,2}$, ..., $X_{l,n}$) be a replication vector for layer l in a replication graph. We define the minimum *activation cost* of vector $\bar{X}_i$ for layer i , $A_i(\bar{X}_i)$, to be the minimum sum of the weights of all possible nodes and edges leading to the selected nodes of layer i in a replication graph. Then the goal of Problem (1) can be achieved by computing the minimal value of $\{A_{\text{sink}}(\bar{X}_{\text{sink}}) + \sum_{p=1}^{n} X_{\text{sink},p} * e_{\text{sink},p}\}$ over all possible values of $\bar{X}_{\text{sink}}$.

3.2 Complexity

In this section, we can show that Problem (1) for a computation-intensive application is NP-complete provided we prove the following:

Lemma 1: For any layer l in the replication graph, the minimum activation cost for two selected nodes $t_{l,p}$ and $t_{l,q}$ will be always greater than that for either node $t_{l,p}$ or $t_{l,q}$ only.

Proof: The Lemma can be proven by contradiction. Let A_1 be the the minimum activation cost for two nodes $t_{l,p}$ and $t_{l,q}$, and A_2 and A_3 be the minimum costs for $t_{l,p}$ and $t_{l,q}$ respectively. Assume that $A_1 < A_2$ and $A_1 < A_3$. Since A_1 includes the activation cost of node $t_{l,p}$, an activation cost for $t_{l,p}$ only can be obtained from A_1. The obtained value c is not necessarily the minimum value for $t_{l,p}$, hence $A_2 \leq c$. The value c is obtained by removing some weighted nodes and edges from replication graph. This implies that $c < A_1$. From above, we find that $A_2 < A_1$, which contradicts the assumption. The same reasoning can be applied to A_3 and reaches a contradiction. Therefore, the assumptions are incorrect and Lemma 1 holds.

□

Lemma 1 can be further extended to the cases where more than two weighted nodes are chosen. The conclusion we can draw is that the more nodes are selected from a layer, the bigger the activation cost is.

Lemma 2: Given a computation-intensive application with its SP task graph $G = (V, E)$ and its assignment graph, if node i has outdegree one and edge $< i, j > \in E$, then for any vector $\bar{X}_j$, the minimal activation cost $A_j(\bar{X}_j)$ can be obtained by choosing only one weighted node from layer i. (i.e. $\sum_{p=1}^{n} X_{i,p} = 1$)

Proof: The Lemma can be proven by contradiction. Since node i has outdegree one and edge

$$< \quad A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X_{i,p}^0 * e_{i,p} + \sum_{q=1}^{n} \min_{X_{i,p}^0=1} (X_{j,q} * \mu_{i,j}(p,q)) = m.$$

The result, $m' < m$, contradicts our assumption. It means that the assumption is wrong and Lemma 2 holds.

□

Lemma 3: Given a computation-intensive application with its SP task graph G, the objective of the minimum cost can be achieved by considering only the replication of the forkers.

Proof: We proceed to prove the lemma by contradiction. Let the minimum cost for task replication problem be z_0 if only the forkers(i.e. outdegree > 1) are allowed to run on more than one processor. Assume the total cost can be reduced further by replicating some task i which is not a forker. Then there are two possible cases for i:

1. i has outdegree 0.

2. i has outdegree 1.

In case 1, i is the sink of the whole graph. Also i may be the joiner of some SP subgraphs. If i is allowed to run on an extra processor b, which is different from the one which i is initially assigned to (when z_0 is obtained), then the new cost will be $z_0 + e_{i,b} + \sum_{<d,i>\in E} \mu_{d,i}$. Apparently, the new cost is greater than z_0. This contradicts our assumption that the total cost can be reduced further by replicating task i.

In case 2, i has one successor. Let $< i,j > \in E$. From the assumption, we know that the replication of i can reduce the total cost. Hence, the minimum activation cost for task instances in layer j, $A_j(\bar{X}_j)$, is obtained when task i is replicated onto more than one processor. This contradicts Lemma 2. Hence, the assumption is incorrect and the objective of the minimum cost can be achieved by considering only the replication of the forkers.

□

Lemma 3 tells that, given an SP graph, if we can find out the optimal replication for the forkers, Problem (1) for computation-intensive applications can be solved. Now, we show that the problem

$< i,j > \in E$, we know that

$$A_j(\bar{X}_j) = \min_{\bar{X}_i}\{A_i(\bar{X}_i) + \sum_{p=1}^{n} X_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))\}.$$

Let us assume that the above equation reaches a minimal value m when more than one node from layer i is selected and the optimal replication vector is $\bar{X}_i^0$. Since $\sum_{p=1}^{n} X_{i,p} > 1$ for $\bar{X}_i^0$, we may remove one selected node from layer i and obtain a new vector $\bar{X}_i'$. Without loss of generality, let us remove $t_{i,r}$. By removing node $t_{i,r}$, a new value m' is obtained. Since m is the minimum value for layer i, it implies that $m \leq m'$.

From Lemma 1, we obtain that $A_i(\bar{X}_i') < A_i(\bar{X}_i^0)$. And for a computation-intensive application, the following holds that $\sum_{q=1}^{n} \mu_{i,j}(p,q) \leq \min_p(e_{i,p})$, $\forall\ 1 \leq p \leq n$. Then,

$$\begin{aligned}
m' &= A_i(\bar{X}_i') + \sum_{p=1}^{n} X'_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q)) \\
&< A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X'_{i,p} * e_{i,p} + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q)) \\
&< A_i(\bar{X}_i^0) + (\sum_{p=1}^{n} X^0_{i,p} * e_{i,p} \ - \ e_{i,r}) + \sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q)) \\
&= A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))] - e_{i,r} \\
&< A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))] - \min_p(e_{i,p}) \\
&\leq A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p} + [\sum_{q=1}^{n} \min_{X'_{i,p}=1}(X_{j,q} * \mu_{i,j}(p,q))] - \sum_{q=1}^{n} \mu_{i,j}(p,q) \\
&< A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X^0_{i,p} * e_{i,p}
\end{aligned}$$

$$< \quad A_i(\bar{X}_i^0) + \sum_{p=1}^{n} X_{i,p}^0 * e_{i,p} + \sum_{q=1}^{n} \min_{X_{i,p}^0=1} (X_{j,q} * \mu_{i,j}(p,q)) = m.$$

The result, $m' < m$, contradicts our assumption. It means that the assumption is wrong and Lemma 2 holds.

□

Lemma 3: Given a computation-intensive application with its SP task graph G, the objective of the minimum cost can be achieved by considering only the replication of the forkers.

Proof: We proceed to prove the lemma by contradiction. Let the minimum cost for task replication problem be z_0 if only the forkers(i.e. outdegree > 1) are allowed to run on more than one processor. Assume the total cost can be reduced further by replicating some task i which is not a forker. Then there are two possible cases for i:

1. i has outdegree 0.

2. i has outdegree 1.

In case 1, i is the sink of the whole graph. Also i may be the joiner of some SP subgraphs. If i is allowed to run on an extra processor b, which is different from the one which i is initially assigned to (when z_0 is obtained), then the new cost will be $z_0 + e_{i,b} + \sum_{<d,i>\in E} \mu_{d,i}$. Apparently, the new cost is greater than z_0. This contradicts our assumption that the total cost can be reduced further by replicating task i.

In case 2, i has one successor. Let $< i,j > \in E$. From the assumption, we know that the replication of i can reduce the total cost. Hence, the minimum activation cost for task instances in layer j, $A_j(\bar{X}_j)$, is obtained when task i is replicated onto more than one processor. This contradicts Lemma 2. Hence, the assumption is incorrect and the objective of the minimum cost can be achieved by considering only the replication of the forkers.

□

Lemma 3 tells that, given an SP graph, if we can find out the optimal replication for the forkers, Problem (1) for computation-intensive applications can be solved. Now, we show that the problem of finding an optimal replication for the forkers in an SP graph is NP-complete. First, a special form of the replication problem is introduced.

Uni-Cost Task Replication (UCTR) problem is stated as follows:

INSTANCE: Graph $G' = (V', E')$, $V' = V_1' \cup V_2'$, where $\mid V_1' \mid = n$ and $\mid V_2' \mid = m$. If $x \in V_1'$ and $y \in V_2'$ then edge $< x, y > \in E'$ (i.e. $\mid E' \mid = m \times n$). For each $x \in V_1'$, there is an activation cost m. Associated with each edge $< x, y > \in E'$, there is a communication cost $d_{x,y} = n \times m$ or 0. A positive integer $K \leq n \times m$ is also given.

QUESTION: Is there a feasible subset $V_k \subseteq V_1'$ such that, we have

$$[\sum_{x \in V_k} m + \sum_{y \in V_2'} \min_{x \in V_k} (d_{x,y})\,] \leq K? \tag{3}$$

[Theorem 1]: Uni-Cost Task Replication problem is NP-Complete.

[Proof]: The problem is in NP because a subset V_k, if it exists, can be checked to see if the sum of activation costs and communication costs is less than or equal to K. We shall now transform the VERTEX COVER [3] problem to this problem. Given any graph $G = (V,E)$ and an integer $C \leq \mid V \mid$, we shall construct a new graph $G' = (V',E')$ and $V' = V_1' \cup V_2'$, such that there exists a VERTEX COVER of size C or less in G if and only if there is a feasible subset of V_1' in G'. Let $\mid V \mid = n$ and $\mid E \mid = m$. To construct G', (1) we create a vertex v_i for each node in V, (2) we number the edges in E, and (3) we create a vertex b_j for each edge $< u,v > \in E$ where u, $v \in V$. We define $K = m \times C$, $V_1' = \{v_1, v_2, \ldots, v_n\}$, $V_2' = \{b_1, b_2, \ldots, b_m\}$ and $E' = \{< v_x, b_y > \mid v_x \in V_1', b_y \in V_2' \}$. Let $d_{v_x,b_y} = 0$, if v_x is an end point of the corresponding edge of vertex b_y; and $= n \times m$, otherwise. An illustration, where $n = 7$ and $m = 9$, is shown in Figure 5.

Let us now argue that there exists a vertex cover of size C or less in G if and only if there is a feasible subset of V_1' in G' to satisfy that the sum of activation cost and communication cost is $m \times C$ or less. Suppose there is a vertex cover of size C, then for each vertex b_y $(= < u,v >)$ in V_2', at least one of u and v belongs to the vertex cover. By selecting all the vertices in the vertex cover into the subset of V_1', we know that the sum in Eq. (3) will be $m \times C$. Since $C \leq n$, it implies that $m \times C \leq n \times m$.

Conversely, for any feasible subset $V_k \subseteq V_1'$ such that the total cost is equal to or less than

mC, we can see that the second term of Eq. (3) (i.e. the sum of communication cost) must be zero. Suppose, for some $g_y \in V_2'$, the minimum communication cost between g_y and vertices in V' is nonzero, then the communication cost will be at least $m \times n$. Since $C \leq n$, it implies that $m \times n \geq m \times C$. The total cost in Eq. (3) will be greater than $m \times C$, which is a contradiction. Thus the minimum communication cost between any vertex in V_2' and any vertex in V_k is zero. It means that at least one of two end points of each edge in E belongs to V_k. Since, there is at most C vertices in V_k (the activation cost for each vertex is m), and by selecting the vertices in V_k, we obtain a vertex cover of size C or less in G.

□

[Theorem 2]: *The problem,* MCRP-SP *for computation-intensive applications, is NP-complete.*

[Proof]: From Lemma 3, we know that only the forker in an SP graph of type T_{and} needs to run on more than one processor. Consider the following recognition version of Problem (1) for SP graphs of type T_{and}:

Given a distributed system of n processors, an SP graph $G^a = (V^a, E^a)$ of type T_{and}, its assignment graph H and two positive integers m and r. Let r be a multiple of m, $V^a = \{s, t, 1, 2, \ldots, r\}$ and $E^c = \{< s,i > \mid i = 1,2,\ldots,r\} \cup \{< i,t > \mid i = 1,2,\ldots,r\}$. Task s (t) is the forker (joiner) of G^a. Execution cost $e_{i,p}$ and communication cost $\mu_{i,j}(p,q)$ are defined in H, $\forall < i,j > \in E^c$ and $\forall\ 1 \leq p,q \leq n$. Integer variable $X_{i,p} = 1$ if task i is assigned to processor p; and $= 0$, otherwise. When a positive integer $K \leq r$ is given, is there an assignment of $X_{i,p}$'s, such that

$$[\sum_{i,p} X_{i,p} * e_{i,p} + \sum_{<i,j> \in E,\ 1 \leq q \leq n} \min_{X_{i,p}=1} (\mu_{i,j}(p,q) * X_{j,q})\ ] \leq K?$$

$$\text{where } \sum_{i,p} X_{i,p} = 1,\ \forall i \neq s, \text{ and } \sum_{i,p} X_{i,p} \geq 1, \text{ if } i = s. \qquad (4)$$

We shall transform the UCTR problem to this problem. Given any graph $G' = (V_1' \cup V_2', E')$ considered in UCTR problem, we construct an SP graph of type T_{and}, $G^c = (V^a, E^a)$, and its assignment graph H, such that G' has a feasible subset of V_1' to allow the sum in Eq. (3) is K or less if and only if there is an assignment of $X_{i,p}$'s for G^a and H to satisfy Eq. (4). Let $\mid V_1' \mid = n$,

| V_2' | = m, then the unit cost $f = n \times m$. Assign $r = m \times f$ (= $n \times m^2$) and $K = n \times m$. The forker and joiner of G^a are s and t respectively. Then $V^a = \{s, t, 1, 2, \ldots, r\}$ and $E^a = \{< s,i > \mid i = 1,2,\ldots,r\} \cup \{< i,t > \mid i = 1,2,\ldots,r\}$. We assign the execution costs and communication costs in H as follows. An illustration, where $m = 2$ and $n = 3$, is shown in Figure 7.

- $\forall\ 1 \le p \le n$, $e_{s,p} = m$.
- $\forall\ 1 \le i \le r$, $\forall\ 1 \le p \le n$, if $p = 1$ then $e_{i,p} = 0$ else $e_{i,p} = r$.
- $\forall\ 1 \le p \le n$, if $p = 1$ then $e_{t,p} = 0$ else $e_{t,p} = r$.
- $\forall\ 1 \le i \le r$, $\forall\ 1 \le p \le n$, let $q = (i-1)$ div $(m \times n)$, where div is the integral division. If $d_{v_p,t_{q+1}} \ne 0$ then $\mu_{s,i}(p,1) = 1$ else $\mu_{s,i}(p,1) = 0$.
- $\forall\ 1 \le i \le r$, $\forall\ 1 \le p \le n$, $\forall\ q \ne 1$, $\mu_{s,i}(p,q) = 0$.
- $\forall\ 1 \le i \le r$, $\forall\ 1 \le p,q \le n$, $\mu_{i,t}(p,q) = 0$.

It is easy to verify that the SP graph constructed by the the above rules is of type T_{and} and computation-intensive. For each node in V_2' of G', we create f nodes in G^a, where the communication cost between each node and source s is either one or zero.

Let us now argue that there exists a feasible subset of V_1' for UCTR problem if and only if there exists a valid assignment of $X_{i,p}$'s such that the total sum in Eq. (4) is K or less. Suppose a feasible subset V_k of V_1' exists such that the sum in Eq. (3) is C ($\le K$) . Let V_1' be $\{v_1,v_2,\ldots,v_n\}$ Then we can obtain a valid assignment by letting $X_{i,1} = 1$, $X_{i,2} = 0, \ldots, X_{i,n} = 0$, $\forall\ 1 \le i \le r$, and $X_{t,1} = 1$, $X_{t,2} = 0, \ldots, X_{t,n} = 0$, and $X_{s,p} = 1$, if $v_p \in V_k$; and $X_{s,p} = 0$, if $v_p \notin V_k$, $\forall\ 1 \le p \le n$. Since each node x in V_2' corresponds to f nodes in G^a, it is sure that the communication cost between node x and any node (v_p) in V_1' is equal to the total communication costs between these f nodes and any task instance of source ($t_{s,p}$) in G^a. By summing up all the costs, we can obtain that the total sum is C. Since $C \le K \le n \times m < r$, this is a valid assignment.

Conversely, if there exists an assignment of $X_{i,p}$'s such that the sum in Eq. (4) is K or less, then the following must be true that $X_{i,1} = 1$, $X_{i,2} = 0, \ldots, X_{i,n} = 0$, $\forall\ 1 \le i \le r$, and $X_{t,1} = 1$, $X_{t,2} = 0, \ldots, X_{t,n} = 0$. It is because for some $p \ne 1$, if $X_{i,p} = 1$ then the sum must be greater than

r, which causes a conflict. Hence the second term in Eq. (4) must be zero. Thus, we may obtain a subset of V_1 for UCTR problem by selecting node $x \in V_1$ if $X_{s,x}$ equals 1. Since the first term in Eq. (3) is equivalent to the first term in Eq. (4), the total sum for UCTR problem will be also K or less then.

□

4 Optimal Replication for SP Graphs of Type T_{and}

In this section, we develop the branch-and-bound algorithm to find an optimal solution for T_{and} subgraphs. The non-forker nodes only need to run on one processor. Hence, an optimal assignment of non-forker nodes can be done after an optimal replication for forkers is obtained.

4.1 A Branch-and-Bound Method for Optimal Replication

Consider a T_{and} SP graph with forker-joiner pair (s,h) shown in Figure 6. There are B subgraphs connected by s and h. These B subgraphs have a parallel-and relationship. Since the joiner h has only one copy in optimal solution (i.e. $\sum_{p=1}^{n} X_{h,p} = 1$), we decompose the minimum cost replication problem $\mathcal{P}$ for a T_{and} SP graph into n subproblems $\mathcal{P}^q$, $q = 1, 2, \ldots, n$, where $\mathcal{P}^q$ is to find the minimum cost when the joiner is assigned to processor q (i.e. $X_{h,q} = 1$).

Given a joiner instance $t_{h,q}$, subgraphs G_b's, $b = 1, 2, \ldots, B$, and the minimum costs $C^b_{p,q}$'s between each forker instance $t_{s,p}$ and joiner instance $t_{h,q}$, $\forall\ 1 \leq p \leq n$ and $1 \leq b \leq B$. we further decompose problem $\mathcal{P}^q$ into n subproblems $\mathcal{P}^q_k$, $k = 1, 2, \ldots, n$, where k is the number of replicated copies that the forker s has. Basically, $\mathcal{P}^q_k$ means the problem of finding an optimal replication for k copies of forker s where the joiner h is assigned to processor q. Since the problem of finding an optimal replication for forker s is NP-complete, we propose a branch-and-bound algorithm for each subproblem $\mathcal{P}^q_k$.

We sort the forker instances according to their execution costs $e_{s,p}$'s into non-decreasing order. Without loss of generality, we assume $e_{s,1} \leq e_{s,2} \leq \ldots \leq e_{s,n}$. We represent all the possible combinations that s may be replicated by a combination tree with $\binom{n}{k}$ leaf nodes. To make the solution efficient, we shall not consider all combinations since it is time-consuming. We apply a least-cost branch-and-bound algorithm to find an optimal solution by traversing a small portion of the combination tree.

During the search, we maintain a variable $\hat{z}$ to record the minimum value known so far. The search is done by the expansion of intermediate nodes. Each intermediate node v at level y represents a combination of y out of n forker instances. The expansion of node v generates at most $n-y$ child nodes, while each child node inherits y forker instances from v and adds one distinct forker instance to itself. For example, if node v is represented by $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y} \succ$, where $i_1 < i_2 < \ldots < i_y$, then $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y}, t_{s,i_y+j} \succ$ represents a possible child node of v, $\forall\ 1 \leq j \leq n-i_y$. A combination tree, where $k = 4$ and $n = 6$, is shown in Figure 8. At any intermediate node of a combination tree, we apply an estimation function to compute the least cost this node can achieve. If the estimated cost is greater than $\hat{z}$, then we prune the node and the further expansion of the node is not necessary. Otherwise, we insert this node along with its estimated cost into a queue. The nodes in the queue are sorted into non-decreasing order of their estimated costs, where the first node of the queue is always the next one to be expanded. When the expansion reaches a leaf node, the actual cost of this leaf is computed. If the cost is less than $\hat{z}$, we update $\hat{z}$. The algorithm terminates when the queue is empty.

4.1.1 The Estimation Function

The proposed branch-and-bound algorithm is characterized by the estimation function. Let node v be at level y of the combination tree associated with subproblem $\mathcal{P}_k^q$ and be represented by $\prec t_{s,i_1}, t_{s,i_2}, \ldots, t_{s,i_y} \succ$, where $i_1 < i_2 < \ldots < i_y$. Any leaf node that can be reached from node v needs $k-y$ more forker instances. Let $\ell = \prec j_1, j_2, \ldots, j_{k-y} \succ$ be a tuple of $k-y$ instances chosen from the remaining $n-i_y$ instances, where $j_1 < j_2 < \ldots < j_{k-y}$. Let L be the set of all possible ℓ's. Let $g(v)$ be the smallest cost among all leaf nodes that can be reached from node v.

$$g(v) = \sum_{a=1}^{y} e_{s,i_a} + \min_{\ell \in L}[\sum_{j_x \in \ell} e_{s,j_x} + \sum_{b=1}^{B} \min_{p=i_1,i_2,\ldots,i_y \text{ or } p\in\ell} (C_{p,q}^{b})\,] + e_{h,q}.$$

Since the complexity involved in computing $g(v)$ is $\binom{n-i_y}{k-y}$, we use the following estimation function $est(v)$ to approximate $g(v)$:

$$est(v) = \sum_{a=1}^{y} e_{s,i_a} + \sum_{j=i_y+1}^{i_y+k-y} e_{s,j} + \sum_{b=1}^{B} \min_{p=i_1,i_2,\ldots,i_y,i_y+1,i_y+2,\ldots,n} (C^b_{p,q}) + e_{h,q}. \tag{5}$$

Since

$$\sum_{j=i_y+1}^{i_y+k-y} e_{s,j} \leq \sum_{j_x \in \ell} e_{s,j_x} \text{ and } \sum_{b=1}^{B} \min_{p=i_y+1,i_y+2,\ldots,n} (C^b_{p,q}) \leq \sum_{b=1}^{B} \min_{p \in \ell} (C^b_{p,q}) ,$$

it is easy to see that $est(v) \leq g(v)$. Hence, we use $est(v)$ as the lower bound of the objective function at node v.

4.1.2 The Proposed Algorithm

Three parameters of the branch-and-bound algorithm are joiner instance ($t_{h,q}$), the number of processors that forker s is allowed to run (k), and the up-to-date minimum cost ($\hat{z}$). The algorithm $BB(k, q, \hat{z})$ is shown in Table 1.

The MCRP-SP problem can be solved by invoking $BB(k, q, \hat{z})$ n^2 times with parameters set to different values. $BB(k, q, \hat{z})$ solves the problem $\mathcal{P}^q_k$, while the whole procedure, shown in Table 2, solves $\mathcal{P}$.

4.2 Performance Evaluation

The essence of the branch-and-bound algorithm is the expansion of the intermediate nodes. Upon the removal of a node from the queue its children are generated and their estimated values are computed. If the estimation function performs well and gives a tight lower bound of objective function, the number of expanded nodes should be small. Then an optimal solution can be found out as soon as possible.

We conduct two sets of experiments to evaluate the performance of the proposed solution. The performance indices we consider are the number of enqueued intermediate nodes (EIM) and the number of visited leaf nodes (VLF) during the search. We calculate EIM and VLF by inserting one counter for each index at lines 13 and 8 of Table 1 respectively. Each time the execution reaches line 13 (8), EIM (VLF) is incremented by 1.

The first set of experiments is on SP graphs of type T_{and} where the communication cost between any two task instances is arbitrary and is generated by random number generator within the range [1,50]. The execution cost for each task instance is also randomly generated within the same range. The second set of experiments is on SP graphs of type T_{and} with the constrain of computation-intensive applications. We vary the size of the problem by assigning different values to the number of processors in the system (n) and the number of parallel-and subgraphs connected by forker and joiner (B). For each size of the problem (n, B), we randomly generate 50 problem instances and solve them. The results, including the average values of EIM and VLF over the solutions of 50 problem instances, are summarized in Table 3.

From Table 3, we find out that the proposed method significantly reduces the number of expansions for intermediate nodes and leaf nodes. For example, for problem size (n, B) = (20, 40), the total number of leaf nodes is 2^{20} (= 1,048,576) if an exhaustive search is applied. However, our algorithm only generates 16,857 nodes on the average, because we apply $est(v)$, $\hat{z}$, and the branch-and-bound approach.

The branch-and-bound approach and the estimation function even perform better for the computation-intensive applications. We can see that EIM and VLF values are much more smaller in Set II than those in Set I. It is because that in the computation-intensive applications an optimal number of replications for the forker is smaller than that in general applications. The $\hat{z}$ value in function $OPT()$ is able to reflect this fact and avoid the unnecessary expansions.

5 Sub-Optimal Replication for SP Graphs of Type T_{and}

The branch-and-bound algorithm in section 4.1 yields an optimal solution for T_{and} subgraphs. However, the complexity involved is in exponential time in the worst case. Hence, we also consider to find a near-optimal solution in polynomial time.

5.1 Approximation Method

For the problem $\mathcal{P}_k^q$ defined in section 4.1, we exploit an approximation approach to solve it in polynomial time. The approach is based on iterative selection in a dynamic programming fashion. Given a joiner instance $t_{h,q}$ and subgraphs G_b, $b = 1, 2, \ldots, B$, and minimum costs $C_{p,q}^b$ between $t_{h,q}$ and $t_{s,p}$, $p = 1, 2, \ldots, n$, and $b = 1, 2, \ldots, B$. we define $Sub(p,b)$ to be the sub-optimal solution for replication of forker s where forker instances $t_{s,1}, t_{s,2}, \ldots, t_{s,p}$ and subgraphs G_1, G_2, $\ldots$, G_b are taken into consideration.

Strategy 1:

$Sub(p,b)$ can be obtained from $Sub(p-1,b)$ by considering one more forker instance $t_{s,p}$. Strategy 1 consists of two steps. The first step is to initialize $Sub(p,b)$ to be $Sub(p-1,b)$ and to determine if $t_{s,p}$ is to be included into $Sub(p,b)$ or not. If yes, then add $t_{s,p}$ in. The second step is to examine if any instances in $Sub(p-1,b)$ should be removed or not. Due to the possible inclusion of $t_{s,p}$ in the first step, we may obtain a lower cost if we remove some instances $t_{s,i}$'s, $i < p$, and reassign the communications for some graphs G_j's from $t_{s,i}$'s to $t_{s,p}$.

Strategy 2:

$Sub(p,b)$ can also be obtained from $Sub(p,b-1)$ by taking one more subgraph G_b into account. Initially, $Sub(p,b)$ is set to be $Sub(p,b-1)$. The first step is to choose the best forker instance from $t_{s,1}, t_{s,2}, \ldots, t_{s,p}$ for G_b. Let the best instance be $t_{s,x}$. The second step is to see if $t_{s,x}$ is in $Sub(p,b)$ or not. If not, a condition is checked to decide whether $t_{s,x}$ should be added in or not. Upon the addition of $t_{s,x}$, we may remove some instances and reassign the communications to achieve a lower cost.

We compare two possible results obtained from the above two strategies and assign the one with lower cost to actual $Sub(p,b)$. Hence by computing in a dynamic programming fashion, $Sub(n,B)$ can be obtained. The algorithm and its graphical interpretation are shown in Figure 9.

5.2 Performance Evaluation

The complexity involved in each strategy described in section 5.1 is $O(nB)$. Since the solving of $Sub(n,B)$ needs to invoke $n \times B$ times of strategies 1 and 2, the total complexity of solving

$Sub(n, B)$ by the approximation method is $O(n^2B^2)$.

We conduct a set of experiments to evaluate the performance of the approximation method. For each problem size (n, B), we randomly generate 50 instances and solve them by using approximation method and exhaustive searching. The data for computation and communication in the experiments are based on the uniform distribution over the range [1,50]. We compare the minimum cost obtained from exhaustive searching (EXHAUST) with those from from approximation (APPROX) and single assignment solution (SINGLE). The optimal single assignment solution is the one in which only one forker instance is allowed. Note that the solutions from SINGLE are obtained from the shortest path algorithm [1]. The results are summarized in Table 4. From the table, we find out that the approximation method yields a tight approximation of the minimum cost. On the contrary, the error range for single copy solution is at least 20%. This again justifies that the replication can lead to a lower cost than an optimal assignment does.

6 Solution of MCRP-SP for computation-intensive applications

6.1 The Solution

Given a computation-intensive application with its SP graph, we generate its parsing tree and assignment graph first. The algorithm finds the minimum weight replication graph from the assignment graph. Then the optimal solution is obtained from the minimum weight replication graph.

The algorithm traverses the parsing tree in the postfix order. Namely, during the traversal, an optimal solution of the subtree S_x, induced by an intermediate node x along with all x's descendant nodes, can be found only after the optimal solutions of x's descendant nodes are found. Given an SP graph G and a distributed system S, we know that there is a one-to-one correspondence between each subtree S_x in a parsing tree $T(G)$ and a limb in the assignment graph of G on S. Whenever a child node b of x is visited, the corresponding limb in the assignment graph will be replaced with a a two-layer T_{chain} limb if b is a T_{chain}- or T_{or}-type node; and a one-layer T_{unit} limb if b is a T_{and}-type node. The algorithm is shown in Table 5. A graphical demonstration of how the algorithm solves the problem is shown in Figure 10.

Before the replacement of a T_{chain} limb is performed (i.e. x is a T_{chain}-type node), each constituent child limb has been replaced with a T_{unit} or two-layer T_{chain} limb. Hence, the shortest path algorithm [1] can be used to compute the weights of the new edges between each node in the source layer and each node in the sink layer of the new T_{chain} limb. The complexity, from lines 05 to 08 of Table 5, in transformation of the limb, corresponding to an intermediate node x with M children, into a two-layer T_{chain} limb is $O(Mn^3)$. An example of illustrating the replacement of a T_{chain} limb is shown from parts (b) to (c) and parts (d) to (e) in Figure 10.

For the replacement of a T_{and} limb, we have to compute $C^b_{p,q}$'s. The values can also be computed by the shortest path algorithm. Hence, the complexity involved in lines 16 and 17 is $O(Bn^3)$. According to the computational model in section 2.2, each task instance s may start its execution if it receives the necessary data from any task instance of its predecessor d. And, from Lemma 2, we know that the minimum sum of initialization costs of multiple task instances of s will be always from only one task instance of d. Therefore, the initialization of task instance $t_{s,p}$ depends on which task instance of d it communicates with. That is why ,in line 19, the communication cost $\mu_{d,s}(i,p)$ is added to the the execution cost of $e_{s,p}$ before $OPT()$ is invoked. And the most significant part of the replacement is to compute the weights on the new edges from the source layer to sink layer. The complexity is $n^2 \times O(OPT())$, which in the worst case is $n^2 2^n$. However, in the average, our OPT function performs pretty well and reduces the complexity significantly. An example of illustrating the replacement of a T_{and} limb is shown from parts (c) to (d) in Figure 10.

We also consider to use the approximation method to find the sub-optimal replacement of a T_{and} limb. In that case, function $OPT()$ in line 21 is replaced with $Sub(n, B)$. The total complexity involved is $O(n^4B^2)$ then.

Finally, for the replacement of a T_{or} limb, if there are B subgraphs connected between the forker and the joiner, then the complexity will be $O(Bn^2)$ for the new edges and $O(Bn^3)$ for $C^b_{p,q}$'s. An example of illustrating the replacement of a T_{or} limb is shown from parts (a) to (b) in Figure 10.

When the traversal reaches the root node of the parsing tree, the result of $FIND()$ will give us either one single layer or two layers, depending on the type of root node. All we have to do is to select the lightest of these n (in single layer) or n^2 (in two layers) shortest path combinations. An optimal replication graph itself is found by combining the shortest paths between the selected nodes that were saved earlier. The whole algorithm has the complexity of

$$O(An^2 2^n) + \sum_i (R_i n^3) + \sum_i (C_i n^3)$$

where A is the number of T_{and} limbs, R_i is the number of subgraphs in the ith T_{or} limb, and C_i is the number of layers in the ith T_{chain} limb. This is not greater than $O(Mn^2 2^n)$, where M is the total number of tasks in the SP graph. The complexity of the algorithm is a linear function of M if the number of processors, n, is fixed.

6.2 Conclusion Remark

This paper has focused on MCRP-SP, the optimal replication problem of SP task graphs for computation-intensive applications. The purpose of replication is to reduce inter-processor communication, and to fully utilize the processor power in the distributed systems. The SP graph model, which is extensively used in modeling applications in distributed systems, is used. The applications considered in this paper are computation-intensive in which the execution cost of a task is greater than its communication cost. We prove that MCRP-SP is NP-complete. We present branch-and-bound and approximation methods for SP graphs of type T_{and}. The numerical results show that the algorithm performs very well and avoids a lot of unnecessary searching. Finally, we present an algorithm to solve the MCRP-SP problem for computation-intensive applications. The proposed optimal solution has the complexity of $O(n^2 2^n M)$ in the worst case, while the approximation solution is in the complexity of $O(n^4 M^2)$, where n is the number of processors in the system and M is the number of tasks in the graph.

For the applications in which the communication cost between two tasks is greater than the execution cost of a task, the replication can still be used to reduce the total cost. However, in the extreme case where the execution cost of each task is zero, the optimal allocation will be to assign each task to one processor. We are studying the optimal replication for the general case.

References

[1] S.H. Bokhari, *Assignment Problems in Parallel and Distributed Computing*, Kluwer Academic Publisheds, MA, 1987.

[2] Y. Chen and T. Chen, "Implementing Fault-Tolerance via Modular Redundancy with Comparison," *IEEE Trans. Reliability*, Vol. 39, pp 217-225, June, 1990.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, San Francisco: W.H. Freeman & Company, Publishers, 1979.

[4] R. Jan, D. Liang and S.K. Tripathi, "A Linear-Time Algorithm for Computing Distributed Task Reliability in Pseudo Two-Terminal Series-Parallel Graphs," *submitted for publication to Journal of Parallel and Distributed Computing.*

[5] C.C. Price and S. Krishnaprasad, "Software Allocation Models for Distributed Computing Systems," in *Proc. 4th International Conference on Distributed Computing Systems*, pp 40-48, May 1984.

[6] D. Liang, A.K. Agrawala, D. Mosse, and Y. Shi, "Designing Fault Tolerant Applications in *Maruti*," *Proc. 3rd International Symposium on Software Reliability Engineering*, pp. 264-273, Research Triangle Park, NC, Oct. 1992.

[7] V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," in *Proc. 4th International Conference on Distributed Computing Systems*, pp 30-39, May 1984.

[8] P.R. Ma, E.Y.S. Lee and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," *IEEE Trans. Computers*, Vol. C-31, pp 41-47, Jan. 1982.

[9] V.F. Magirou and J.Z. Milis, "An Algorithm for the Multiprocessor Assignment Problem," *Operations Research Letters*, Vol. 8, pp 351-356, Dec. 1989.

[10] C.C. Price and U.W. Pooch, "Search Techniques for a Nonlinear Multiprocessor Scheduling Problem," *Naval Res. Logist. Quart.*, Vol 29, pp 213-233, June 1982.

[11] H.S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithm," *IEEE Trans. Soft. Eng.* Vol 3, pp 85-93, Jan. 1977.

[12] D. Towsley, "Allocating Programs Containing Branches and Loops Within a Multiple Processor System," *IEEE Trans. Software Eng.*, Vol. SE-12, pp. 1018-1024, Oct, 1986.

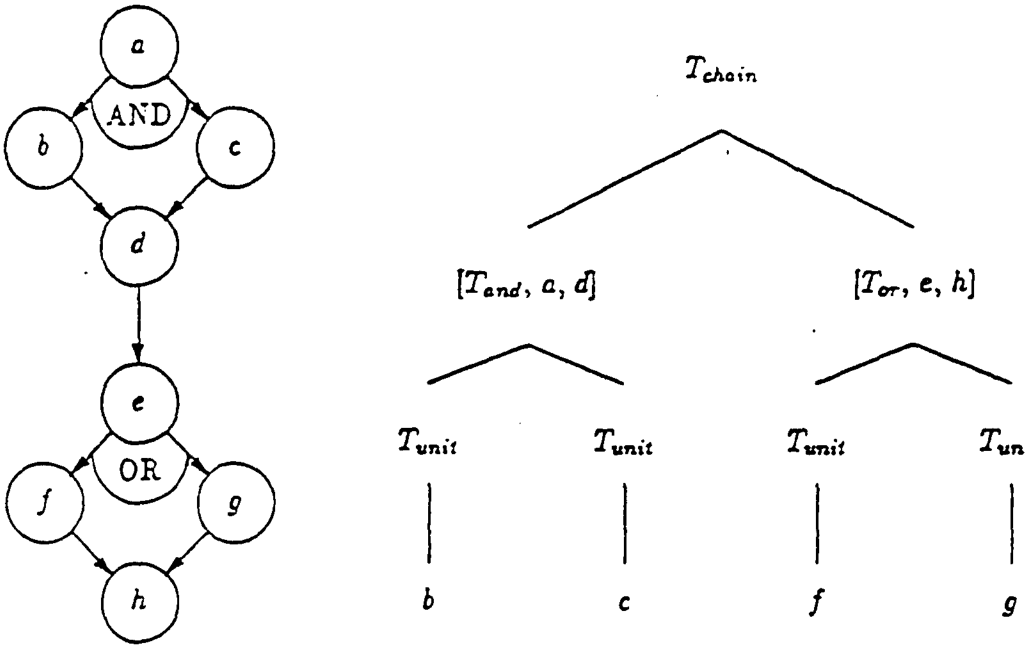
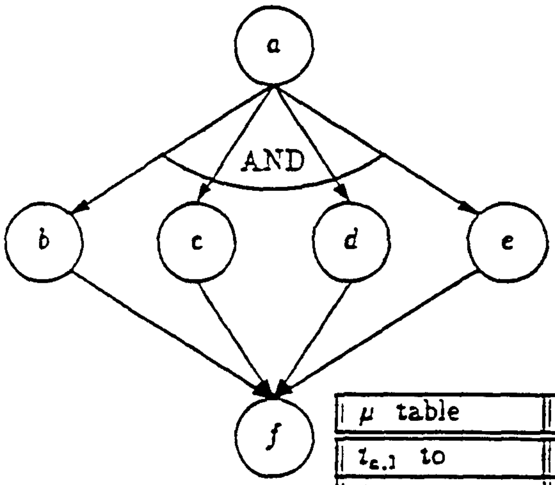


Figure 1: An SP graph and its parsing tree



e table	processor 1	processor 2
task a on	5	5
task b on	7	16
task c on	10	20
task d on	25	8
task e on	14	6
task f on	10	13

μ table	$t_{b,1}$	$t_{b,2}$	$t_{c,1}$	$t_{c,2}$	$t_{d,1}$	$t_{d,2}$	$t_{e,1}$	$t_{e,2}$
$t_{a,1}$ to	1	4	1	4	1	4	1	4
$t_{a,2}$ to	4	1	4	1	4	1	4	1
to $t_{f,1}$ from	3	3	3	3	3	3	3	3
to $t_{f,2}$ from	3	3	3	3	3	3	3	3

Optimal Assignment:

$$e_{a,1} + \mu_{a,b}(1,1) + \mu_{a,c}(1,1) + \mu_{a,d}(1,2) + \mu_{a,e}(1,2) + e_{b,1} + e_{c,1}$$
$$+ e_{d,2} + e_{e,2} + \mu_{b,f}(1,1) + \mu_{c,f}(1,1) + \mu_{d,f}(2,1) + \mu_{e,f}(2,1) + e_{f,1} = 68$$

Optimal Replication:

$$e_{a,1} + e_{a,2} + \mu_{a,b}(1,1) + \mu_{a,c}(1,1) + \mu_{a,d}(2,2) + \mu_{a,e}(2,2) + e_{b,1}$$
$$+ e_{c,1} + e_{d,2} + e_{e,2} + \mu_{b,f}(1,1) + \mu_{c,f}(1,1) + \mu_{d,f}(2,1) + \mu_{e,f}(2,1) + e_{f,1} = 67$$

Figure 2: An example to show how the replication can reduce the total cost

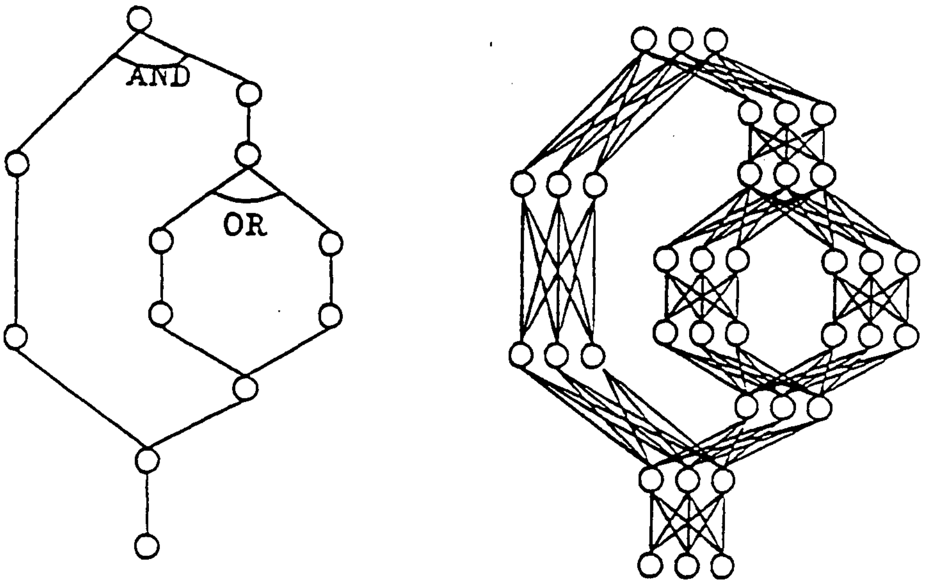


Figure 3: An SP graph and its assignment graph.

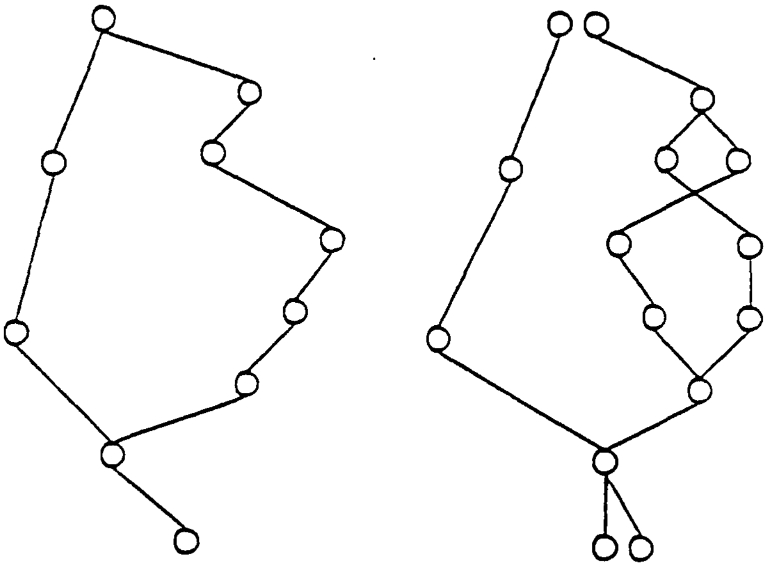


Figure 4: An allocation graph and a replication graph of Figure 3.

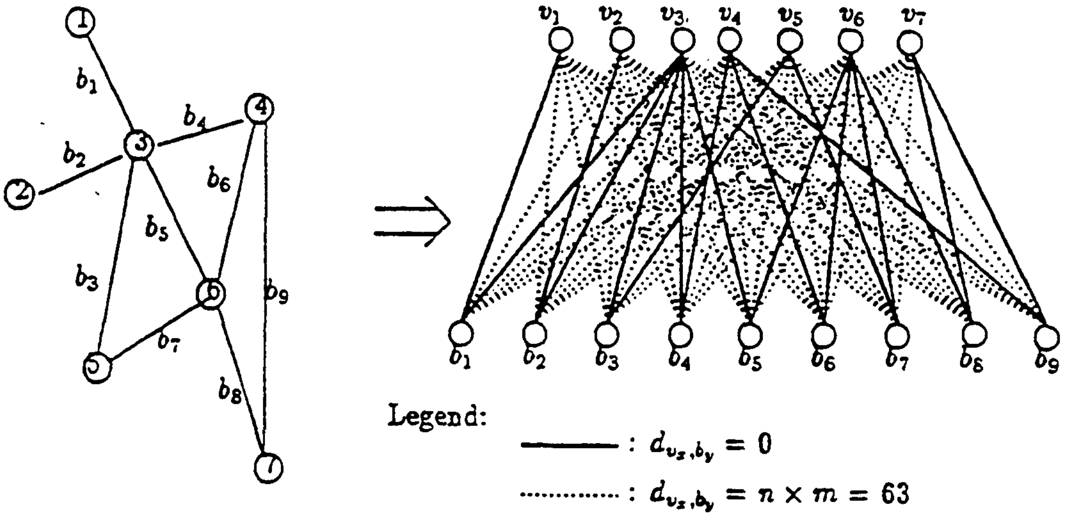

1
2
3
4
5
6
7
b_1
b_2
b_3
b_4
b_5
b_6
b_7
b_8
b_9
v_1 v_2 v_3 v_4 v_5 v_6 v_7
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
Legend:
: $d_{v_x,b_y} = 0$
: $d_{v_x,b_y} = n \times m = 63$

Figure 5: An illustration about how to transform a graph to a UCTR instnace

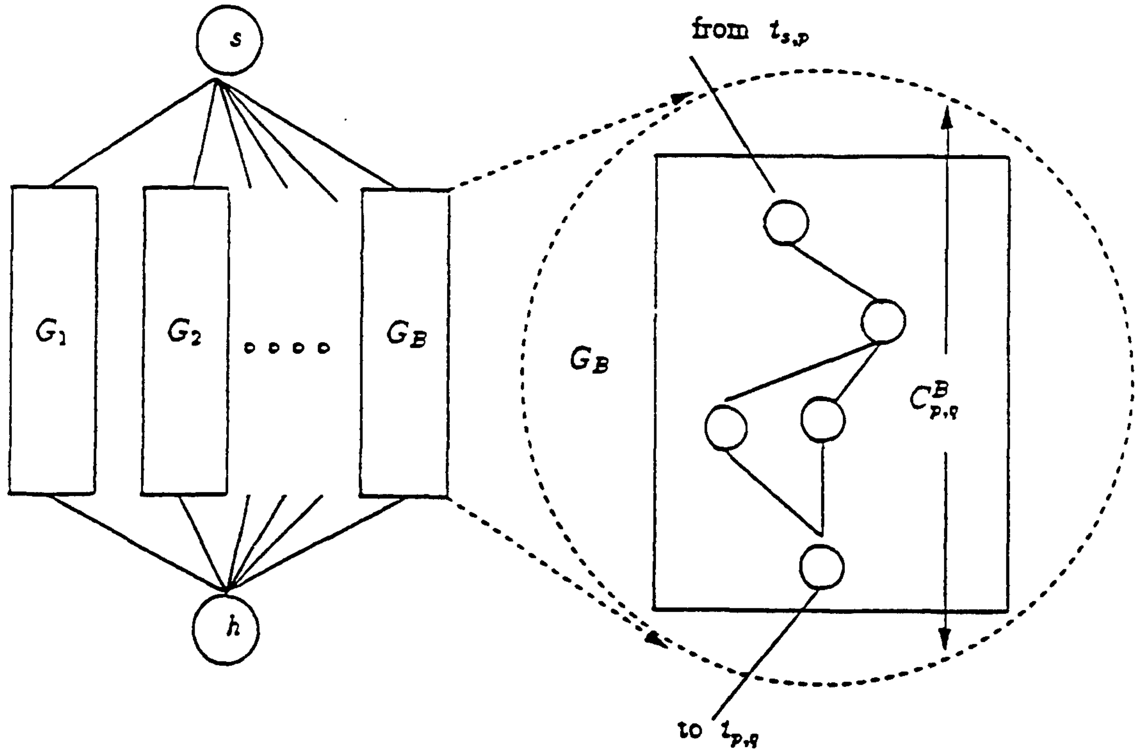
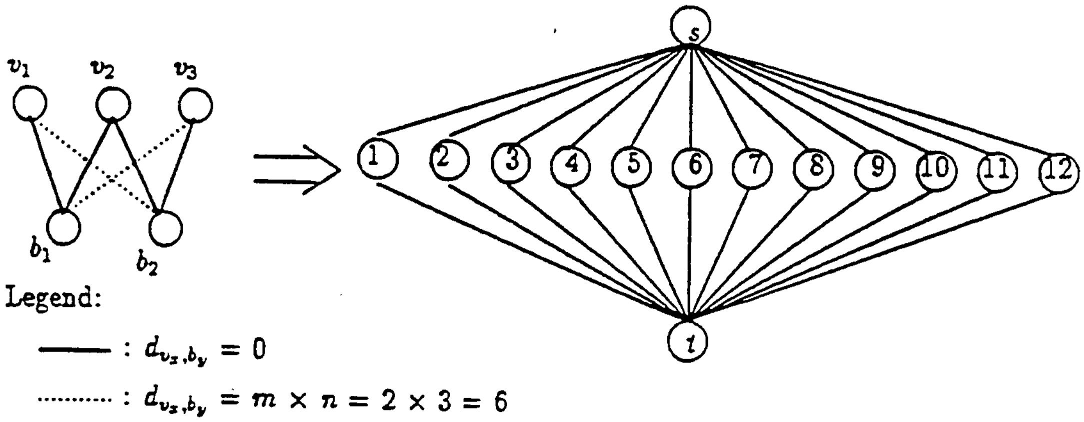


Figure 6: A T_{and} SP graph and the graphical intertrepation of $C^b_{p,q}$.



e table	$p = 1$	$p = 2$	$p = 3$
$e_{s,p} =$	2	2	2
$e_{1,p} =$	0	12	12
$e_{2,p} =$	0	12	12
$e_{3,p} =$	0	12	12
$e_{4,p} =$	0	12	12
$e_{5,p} =$	0	12	12
$e_{6,p} =$	0	12	12
$e_{7,p} =$	0	12	12
$e_{8,p} =$	0	12	12
$e_{9,p} =$	0	12	12
$e_{10,p} =$	0	12	12
$e_{11,p} =$	0	12	12
$e_{12,p} =$	0	12	12
$e_{t,p} =$	0	12	12

μ table	$p = 1$	$p = 2$	$p = 3$
$\mu_{s,1}(p,1) =$	0	0	1
$\mu_{s,2}(p,1) =$	0	0	1
$\mu_{s,3}(p,1) =$	0	0	1
$\mu_{s,4}(p,1) =$	0	0	1
$\mu_{s,5}(p,1) =$	0	0	1
$\mu_{s,6}(p,1) =$	0	0	1
$\mu_{s,7}(p,1) =$	1	0	0
$\mu_{s,8}(p,1) =$	1	0	0
$\mu_{s,9}(p,1) =$	1	0	0
$\mu_{s,10}(p,1) =$	1	0	0
$\mu_{s,11}(p,1) =$	1	0	0
$\mu_{s,12}(p,1) =$	1	0	0
$\mu_{s,i}(p,q) = 0, \forall 1 \leq i \leq 12, p = 1,2,3$ and $q = 2,3$			
$\mu_{i,t}(p,q) = 0, \forall 1 \leq i \leq 12, \forall 1 \leq p, q \leq 3$			

Figure 7: An illustration about how to transform a UCTR instance to a T_{and} SP graph

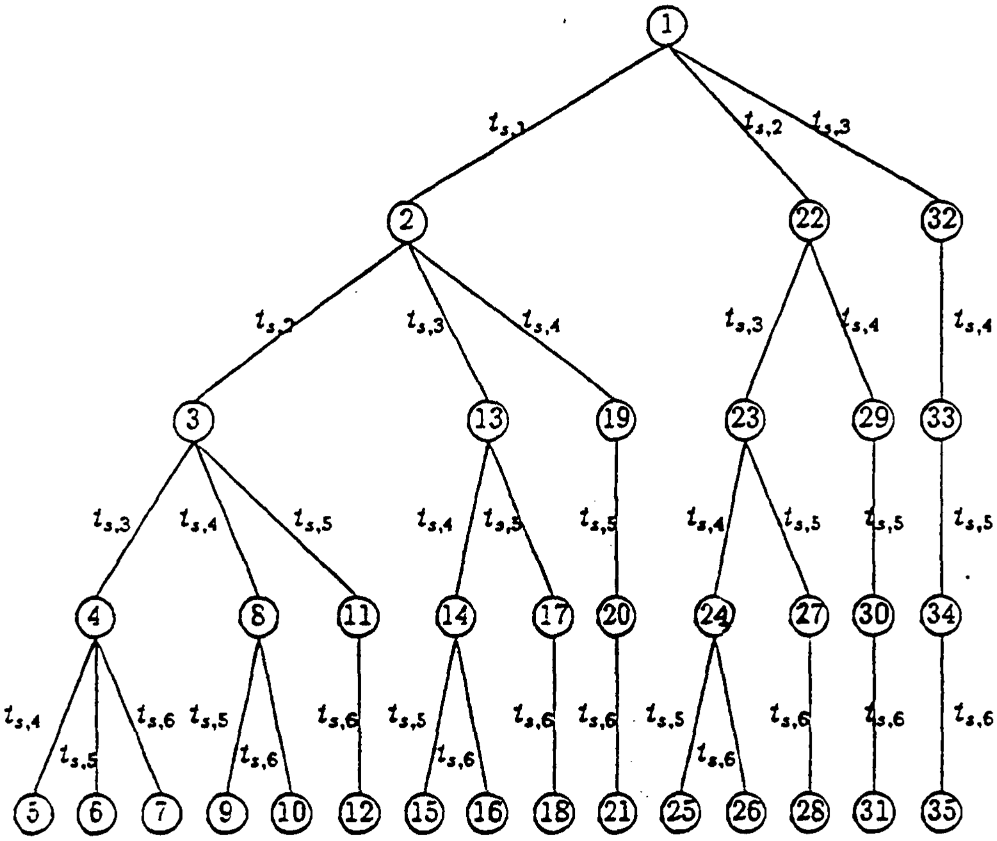


Figure 8: A combination tree for the case where $k = 4$ and $n = 6$

Table 1: Function $BB(k, q, \hat{z})$: branch-and-bound algorithm for solving problem $\mathcal{P}_k^q$

```
Initialize the queue to be empty;
Insert root node $v_0$ into the queue;
While the queue is not empty do begin
    Remove the first node $u$ from the queue;
    Generate all child nodes of $u$ ;
    For each generated child node $v$ do begin
        If $v$ is a leaf node (i.e. $v$ is at level $k$) then
            Compute $g(v)$ by setting $L$ to be $\phi$ ;
            Set $\hat{z} =$ min ( $\hat{z}$, $g(v)$);
        else begin /* $v$ is an intermediate node */
            Compute $est(v)$ by (5) ;
            If $est(v) < \hat{z}$ then
                Insert $v$ into the queue according to $est(v)$ ;
        end;
    end;
end;
Return($\hat{z}$).
```

Table 2: Function $OPT(C_{p,q}^b{'}s,\ e_{s,p}{'}s)$: the optimal solution of MCRP-SP of type T_{and} when $C_{p,q}^b$'s and $\epsilon_{s,p}$'s are given

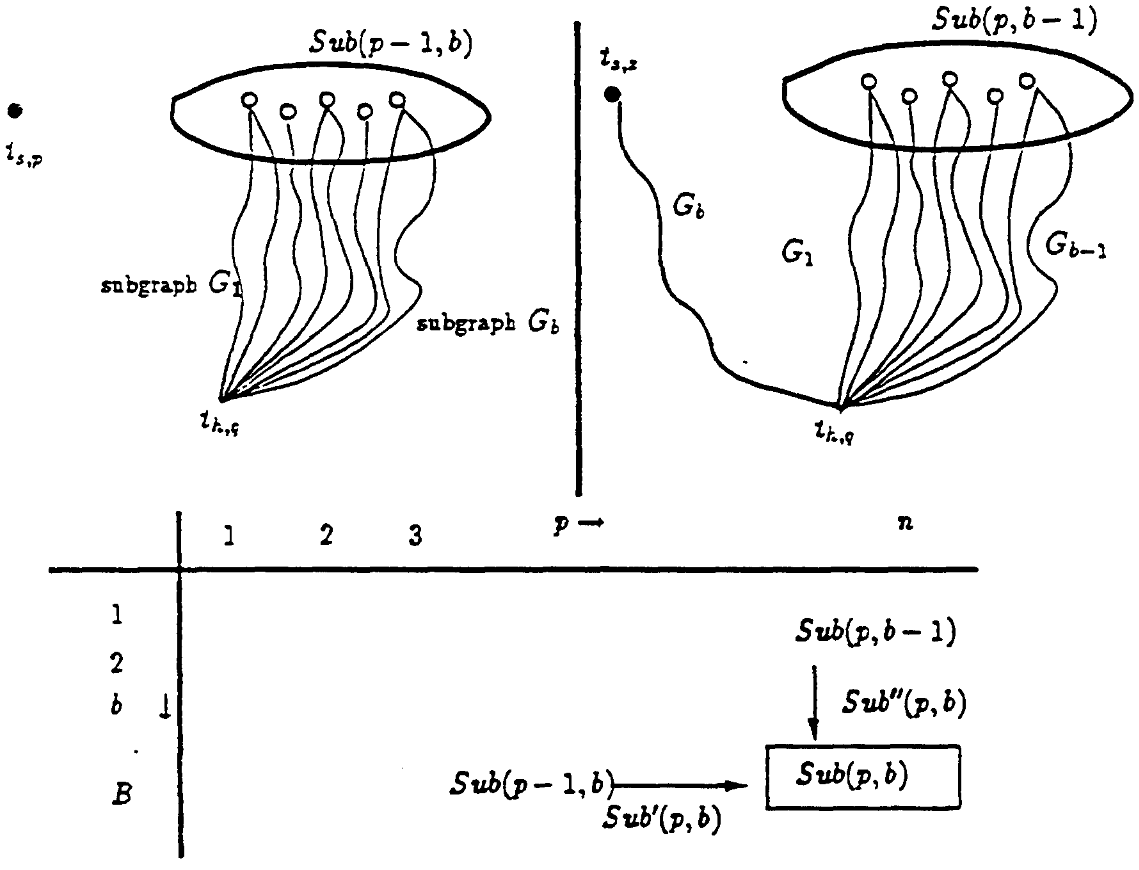
```
Sort $t_{s,p}$'s into a non-decreasing order by values of $\epsilon_{s,p}$'s ;
For $q = 1$ to $n$ do begin
    Let node $v$ be a leaf node at level 1;
    Set $v$ to be $t_{s,1}$ and $k$ to be 1;
    Compute $g(v)$ by setting $L$ to be $\phi$ ;
    Initialize $\hat{z}$ to be $g(v)$ ;
    For $k = 1$ to $n$ do
        $\hat{z} = BB(k, q, \hat{z})$ ;
    Set $c(q) = \hat{z}$ ;
end;
Output the combination with the minimum value among $c(1), c(2), \ldots, c(n)$.
```

Figure 9: Pseudo code, graphical demonstration, and dynamic programming table for approximation methods

```
$Sub(p-1,b) \rightarrow Sub'(p,b)$:
If $e_{s,p} \le \sum_{i=1}^{b}([\min_{x \in Sub(p-1,b)}(C^i_{x,q})] - C^i_{p,q})^+$
begin
    $Sub'(p,b) = Sub(p-1,b) \oplus t_{s,p}$
    Reassign&Remove($Sub'(p,b)$)
end
Else $Sub'(p,b) = Sub(p-1,b)$

Legend:
    $(x)^+ = x$, if $x > 0$.
    $(x)^+ = 0$, if $x \le 0$.
```

```
$Sub(p,b-1) \rightarrow Sub''(p,b)$:
Let $t_{s,x}$ be the one satisfys $\min_{1 \le i \le p}(C^b_{i,q})$ .
If $t_{s,x} \in Sub(p,b-1)$ then
    $Sub''(p,b) = Sub(p,b-1)$
Else
if $e_{s,x} \le \sum_{i=1}^{b}([\min_{j \in Sub(p,b-1)}(C^i_{j,q})] - C^i_{x,q})^+$
begin
    $Sub''(p,b) = Sub(p-1,b) \oplus t_{s,x}$
    Reassign&Remove($Sub''(p,b)$)
end
Else $Sub''(p,b) = Sub(p,b-1)$
```



$$Sub(p,b) = Min\_Cost(Sub'(p,b), Sub''(p,b))$$

Table 3: Computation Results for branch-and-bound approach

		Set I		Set II		Total Number of
n	B	EIM‡	VLF‡	EIM‡	VLF‡	leaves (2^n)
4	20	2	6	4	7	16
	24	3	6	3	6	16
	28	4	7	3	6	16
	32	4	7	3	6	16
	36	4	7	4	7	16
	40	3	6	3	6	16
8	20	36	74	16	51	256
	24	40	75	21	62	256
	28	50	86	26	68	256
	32	63	94	37	78	256
	36	73	96	47	84	256
	40	81	97	50	86	256
12	20	186	558	81	340	4,096
	24	231	639	102	398	4,096
	28	349	839	167	543	4,096
	32	451	967	204	617	4,096
	36	454	984	269	720	4,096
	40	636	1,186	301	780	4,096
16	20	758	3,216	203	1,175	65,536
	24	1,065	4,161	329	1,711	65,536
	28	1,335	4,862	546	2,496	65,536
	32	1,884	6,250	726	3,127	65,536
	36	2,322	7,227	839	3,493	65,536
	40	2,880	8,511	1,179	4,510	65,536
20	20	2,026	12,042	389	3,079	1,048,576
	24	3,579	18,866	761	5,280	1,048,576
	28	5,551	27,018	1,227	7,905	1,048,576
	32	6,405	30,521	1,709	10,357	1,048,576
	36	9,517	40,767	2,681	15,032	1,048,576
	40	11,651	48,087	3,086	16,857	1,048,576

‡: Each value shown is the average value over 50 runs.

Table 4: Simulation Results for Approximation Method

n	B	SINGLE‡	APPROX‡	EXHAUST‡	single error %	approx error %
4	20	2876	2407	2400	20	0.28
	24	3463	2835	2831	22	0.16
	28	4032	3264	3259	24	0.18
	32	4606	3678	3673	25	0.11
	36	5198	4084	4082	27	0.05
	40	5790	4514	4514	28	0.00
8	20	2794	2282	2250	24	1.46
	24	3356	2672	2636	27	1.38
	28	3931	3060	3028	30	1.05
	32	4540	3443	3413	33	0.88
	36	5127	3831	3800	35	0.80
	40	5683	4215	4192	36	0.55
12	20	2767	2213	2161	28	2.42
	24	3359	2592	2542	32	1.99
	28	3912	2996	2941	33	1.88
	32	4491	3364	3299	36	1.97
	36	5063	3736	3676	38	1.62
	40	5610	4101	4043	39	1.43
16	20	2733	2167	2111	29	2.66
	24	3287	2558	2492	32	2.66
	28	3844	2932	2865	34	2.31
	32	4393	3315	3240	36	2.32
	36	4991	3659	3584	39	2.10
	40	5558	4045	3970	40	1.89

‡: Each value shown is the average value over 50 runs.

$$\text{single error\%} = \frac{\text{SINGLE} - \text{EXHAUST}}{\text{EXHAUST}} \times 100\%.$$

$$\text{approx error\%} = \frac{\text{APPROX} - \text{EXHAUST}}{\text{EXHAUST}} \times 100\%.$$

Table 5: Algorithm $FIND(S_x)$: the algorithm for finding the shortest path combinations from the limb which corresponds to the subtree S_x induced by an intermediate node x and all x's descendant nodes in a parsing tree

```
01 Case of the type of intermediate node $x$:
02   Type $T_{chain}$ :
03       For $b$ = the $first$ child node of $x$ to the $last$ one do
04           $FIND(S_b)$; /* Now the limb corresponding to $S_b$ is replaced */
05       Replace the limb corresponding to $S_x$ with a two-layer $T_{chain}$ limb where
06       the source (sink) layer of the old limb is the source (sink) layer of new 2-layer limb;
07       Put weights on the edges between source and sink layers equal to the shortest path
08       between the corresponding nodes;
09
10   Type $T_{and}$ : /* Let x = [ $T_{and}$, forker $s$, joiner $h$ ] */
11       Let $d$ be the predecessor of forker $s$ in $G$ (i.e. $< d,s > \in V$);
12       Let $B$ be the number of child nodes of $x$ in the parsing tree;
13        /* I.e. there are $B$ subgraphs connected by $s$ and $h$ */
14       For $b$ = the first child node of $x$ to the $B$-th child of $x$ do
15           $FIND(S_b)$; /* Now the limb corresponding to $S_b$ is replaced */
16       For $p = 1$ to $n$, $q = 1$ to $n$ and $b = 1$ to $B$ do
17           Compute the minimum replication cost $C^b_{p,q}$ from $t_{s,p}$ to $t_{h,q}$ w.r.t. child $b$ ;
18       For $i = 1$ to $n$ do begin
19           For $p = 1$ to $n$ do  $E_{s,p} = \mu_{d,s}(i,p) + e_{s,p}$ ;
20           /* $E_{s,p}$ accounts for initialization by $t_{d,i}$ and execution cost itself. */
21           For $q = 1$ to $n$ do  $\mu_{d,h}(i,q) = OPT(C^b_{p,q}{}'s, E_{s,p}{}'s)$ ;
22           /* Create new edges from $t_{d,i}$'s to $t_{h,q}$'s */
23       end;
24       Replace the $T_{and}$ limb with a $T_{unit}$ limb, where source layer = sink layer = layer $h$,
25       and there are new edges from layer $d$ to layer $h$;
26
27   Type $T_{or}$ : /* Let x = [ $T_{or}$, forker $s$, joiner $h$ ] */
28       Use the same method described above from lines 12 to 17 to compute $C^b_{p,q}$'s ;
29       Replace the $T_{or}$ limb with a two-layer $T_{chain}$ limb, where
30       the source (sink) layer of $T_{or}$ limb is the source (sink) layer of $T_{chain}$ limb and
31       $\mu_{s,h}(p,q) = \min_b(C^b_{p,q})$, $\forall$ $p$ and $q$ ;
32 end case;
33 Save the shortest paths between any node in source layer and any node
   in sink layer for future reference.
```

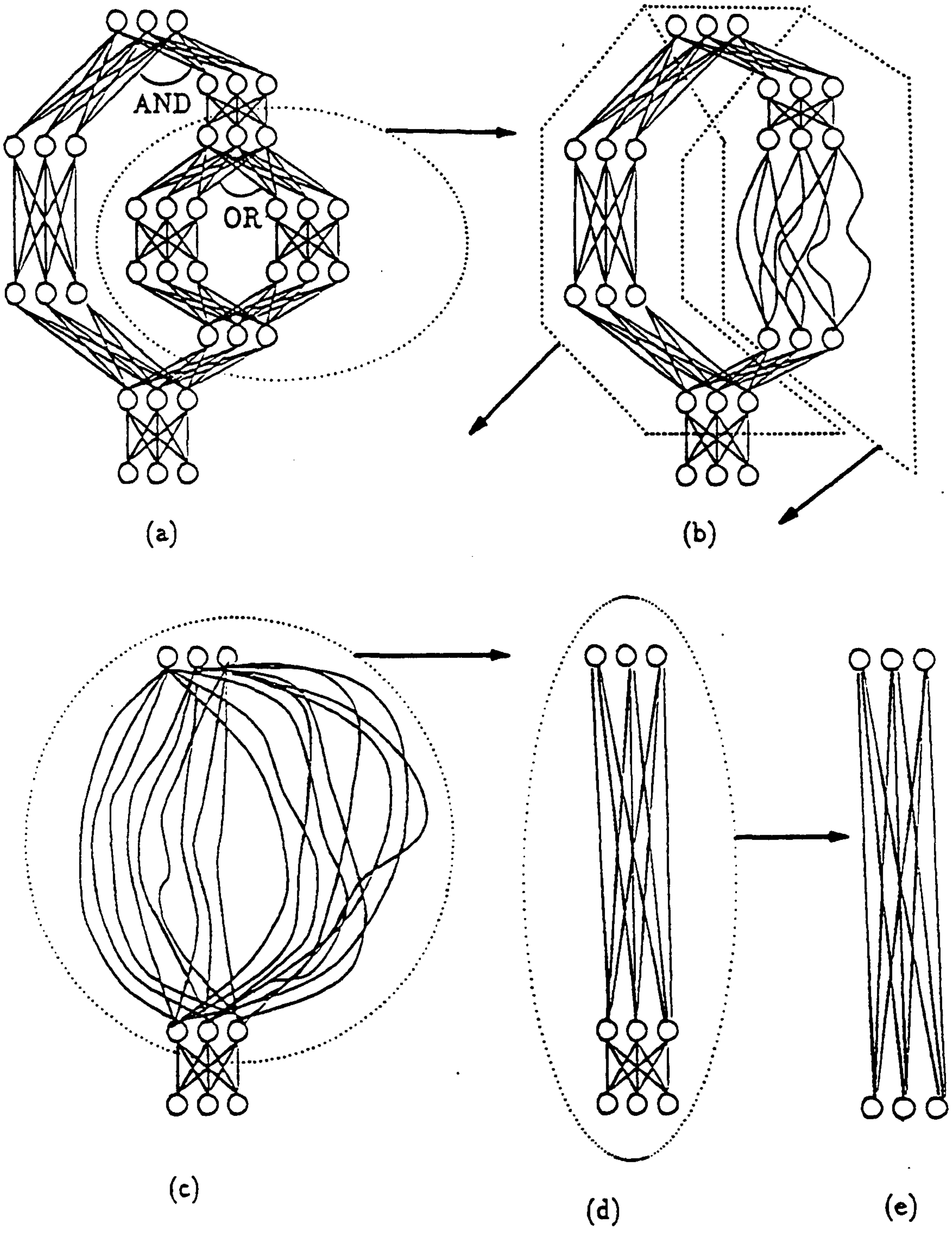


Figure 10. A graphical demostration of how to find an optimal solution for MCRP-SP

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 10/12/94	3. REPORT TYPE AND DATES COVERED Technical

4. TITLE AND SUBTITLE

Optimal Replication of Series-Parallel Graphs for Computation-Intensive Applications Revised Version

5. FUNDING NUMBERS

N00014-91-C-0195
DASG-60-92-C-0055

6. AUTHOR(S)

Sheng-Tzong Cheng and Ashok K. Agrawala

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Computer Science
A. V. Williams Building
University of Maryland
College Park, MD 20742

8. PERFORMING ORGANIZATION REPORT NUMBER

Revised Version
CS-TR-3020.1
UMIACS-TR-93-4.1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Honeywell, Inc.
3600 Technology Drive
Minneapolis, MN 55418

Phillips Laboratory
Directorate of Contracting
3651 Lowry Avenue SE
Kirtland AFB NM 87117-5777

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

This version supercedes the previous version.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

We consider the replication problem of series-parallel(SP) task graphs where each task may run on more than one processor. The objective of the problem is to minimize the total cost of task execution and interprocessor communication. We call it, the minimum cost replication problem for SP graphs (MCRP-SP). In this paper, we adopt a new communication model where the purpose of replication is to reduce the total cost. The class of applications we consider is computation-intensive applicati in which the execution cost of a task is greater than its communication cost. The complexity of MCRP-SP for such applications is proved to be NP-complete. We present a branch-and-bound method to find an optimal solution as well as an approximation approach for suboptimal solution. The numerical results show that such replication may lead to a lower cost than the optimal assignment problem (in which each task is assigned to only one processor) does. The proposed optimal solution has the complexi of $O(n^2 2^n M)$, while the approximation solution has $O(n^4 M^2)$, where n is the number of processors in the system and M is the number of tasks in the graph.

14. SUBJECT TERMS Operating Systems Storage Management, Communications Management	15. NUMBER OF PAGES 35 pages
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclasssified	Unclassified	Unclassified	Unlimited

Designing Temporal Controls [*][†]

Ashok K. Agrawala Seonho Choi
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742
{agrawala, seonho}@cs.umd.edu

Leyuan Shi
Department of Industrial Engineering
University of Wisconsin
Madison, WI 53706
leyuan@ie.engr.wisc.edu

Abstract

Traditional control systems have been designed to exercise control at regularly spaced time instants. When a discrete version of the system dynamics is used, a constant sampling interval is assumed and a new control value is calculated and exercised at each time instant. In this paper we formulate a new control scheme, *temporal control*, in which we not only calculate the control value but also decide the time instants when the new values are to be used. Taking a discrete, linear, time-invariant system, and a cost function which reflects a cost for computation of the control values, as an example, we show the feasibility of using this scheme. We formulate the temporal control scheme as a feedback scheme and, through a numerical example, demonstrate the significant reduction in cost through the use of temporal control.

[*] This work is supported in part by ONR and DARPA under contract N00014-91-C-0195 to Honeywell and Computer Science Department at the University of Maryland. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, ONR, the U.S. Government or Honeywell. Computer facilities were provided in part by NSF grant CCR-8811954.

[†] This work is supported in part by ARPA and Philips Labs under contract DASG-60-92-C-0055 to Department of Computer Science, University of Maryland. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, PL, or the U.S. Government.

1 Introduction

Control systems have been used for the control of dynamic systems by generating and exercising control signals. Traditional approach for feedback controls has been to define the control signals, $u(t)$, as a function of the current state of the system, $x(t)$. As the state of the system changes continuously the controls change continuously, i.e. they are defined as functions of time, t, such that time is treated as a continuous variable. When computers are used for implementing the control systems, due to the discrete nature of computations, time is treated as a discrete variable obtained by regularly spaced sampling of the time axis at Δ seconds. Many standard control formulations are defined for the discrete version of the system, with system dynamics expressed at discrete time instants. In these formulations the system dynamics and the control are expressed as sequences, $x(k)$ and $u(k)$.

Most of the traditional control systems were designed for dedicated controllers which had only one function, to accept the state values, $x(k)$ and generate the control, $u(k)$. However, when a general purpose computer is used as a controller, it has the capabilities, and may, therefore, be used for other functions. Thus, it may be desirable to take into account the cost of computations and consider control laws which do not compute the new value of the control at every instant. When no control is to be exercised, the computer may be used for other functions. In this paper we formulate such a control law and show how it can be used for control of systems, achieving the same degree of control as traditional control systems while reducing computation costs by changing the control at a few, specific time instants. We term this *temporal control.*

To the best of our knowledge this approach to the design and implementation of controls has not been studied in the past. However, taking computation time delay into consideration for real-time computer control has been studied in several research papers [1, 5, 6, 9, 11, 13]. But, all of these papers concentrated on examining computation time delay effects and compensating them while maintaining the assumption of exercising controls at regularly spaced time instants.

The basic idea of temporal control is to determine not only the values for u but also the time instants at which the values are to be calculated and changed. The control values are assumed to remain constant between changes. By exercising control over the time instants of changes the designer has an additional degree of freedom for optimization. In this paper we present the idea and demonstrate its feasibility through an example using a discrete, linear, and time invariant system. Clearly, the same idea can be extended to continuous time as well as non-linear system.

The paper is organized as follows. In Section 2, we formulate the temporal control problem and introduce computation cost into performance index function. The solution approach for temporal control scheme is discussed in *Section 3.* In Section 4, implementation issues are addressed. We provide an example of controlling rigid body satellite in Section 5 . In this example, an optimal temporal controller is designed. Results show that the temporal control approach performs better than the traditional sampled data control approach with the same number of control exercises. Section 6 deals with the application of temporal controls to the design of real-time control systems. Finally, Section 7, we present our conclusions.

2 Problem Formulation

In temporal control, the number of control changes and their exercising time instants within the controlling interval $[0, T_f]$ is decided to minimize a cost function. To formulate the temporal control problem for a discrete, linear time-invariant system, we first discretize the time interval $[0, T_f]$ into M subintervals of length $\Delta = T_f/M$. Let $D_M = \{0, \Delta, 2\Delta, \ldots, (M-1)\Delta\}$ which denote M time instants which are regularly spaced. Here, control exercising time instants are restricted within D_M for the purpose of simplicity. The linear time-invariant controlled process is described by the difference equation:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{1}$$

where k is the time index. One unit of time represents the subinterval Δ, whereas $x \in \mathcal{R}^n$ and $u \in \mathcal{R}^l$ are the state and input vectors respectively.

It is well known that there exists an optimal control law [4]

$$u^o(i) = f[x(i)] \quad i = 0, 1, ..., M-1 \tag{2}$$

that minimizes the quadratic performance index function (Cost)

$$J_M = \sum_{k=0}^{M-1} [x^T(k)Qx(k) + u^T(k)Ru(k)] + x^T(M)Qx(M) \tag{3}$$

where $Q \in \mathcal{R}^{nxn}$ is positive semi-definite and $R \in \mathcal{R}^{lxl}$ is positive definite.

As we can see, traditional controller exercises control at every time instant in D_M. However, in temporal control, we are no longer constrained to exercise control at every time instant in D_M. Therefore, we want to find an optimal control law, δ and g for $i = 0, 1, ..., M-1$:

$$\begin{aligned} u^o(i) &= u^o(i-1) \quad if\ \delta(i) = 0 \\ u^o(i) &= g[x(i)] \quad if\ \delta(i) = 1 \end{aligned} \tag{4}$$

that minimizes a new performance index function

$$\begin{aligned} J'_M &= \sum_{k=0}^{M-1}[x^T(k)Qx(k)+u^T(k)Ru(k)]+x^T(M)Qx(M)+\sum_{k=0}^{M-1}\delta(k)\mu \qquad (5) \\ &= J_M + C_M \end{aligned}$$

Here, μ is the computation cost of getting a new control value at a time instant, and $C_M = \sum_{k=0}^{M-1}\delta(k)\mu$ denotes the total computation cost. Note that $\nu = \sum_{k=0}^{M-1}\delta(k)$ is the number of control changes. Also, let $D_\nu = \{t_0, t_1, t_2, \ldots, t_{\nu-1}\}$ consist of control changing time instants where $t_0 = 0$, $t_1 = n_1\Delta$, ..., $t_{\nu-1} = n_{\nu-1}\Delta$. That is, $n_0, n_1, n_2, \ldots, n_{\nu-1}$ are the indices for control changing time instants and $\delta(n_i) = 1$ for $i = 0, 1, 2, \ldots \nu - 1$.

With this new setting we need to choose ν, D_ν, and control input values to find an optimal controller which minimizes J'_M. This new cost function is different from J_M in two aspects. First, the concept of computational cost is introduced in J'_M as C_M term to regulate the number of control changes chosen. If we do not take this computation cost into consideration ν is likely to become M. If computation cost is high (i.e., μ has a large value) then ν is likely to be small in order to minimize the total cost function. Second, in temporal control, not only do we seek optimal control law $u(x(t))$, but also the control exercising time instants and the number of control changes. In the next section, we present in detail specific techniques for finding an optimal temporal control law.

3 Temporal Control

We develop a three-step procedure for finding an optimal temporal controller.

Step 1. Find an optimal control law given ν and D_ν

Step 2. Find best D_ν given ν

Step 3. Find best ν

First, in the following two subsections(3.1 and 3.2) we derive a temporal control law which minimizes the cost function J'_M when D_ν is given, i.e., both time instants and number of controls are fixed. Since ν and D_ν are fixed we can use J_M defined in (5) as a cost function instead of J'_M. Secondly, assume that ν is fixed but D_ν can vary. Then we present an algorithm in section 3.3 to find a D^o_ν such that J_M (and J'_M) is minimized. Finally, we will vary ν from 1 to ν_{max} to search an optimal D^o_ν at which temporal control should be exercised. Section 3.4 presents this iteration procedure. Section 3.5 explains how to incorporate terminal state constraints into the above procedure of getting an optimal temporal control law. And a complete algorithm of the above procedure is described in Section 3.6. Finally, in Section 3.7 we explain how to get optimal temporal controllers over an initial state space.

3.1 Closed-loop Temporal Control with D_ν Given

Assume that ν and D_ν are given. Then a new control input calculated at t_i will be applied to the actuator for the next time interval from t_i to t_{i+1}. Our objective here is to determine the optimal control law

$$u^o(n_i) = g[x(n_i)] \quad i = 0, 1, ..., \nu - 1 \tag{6}$$

that minimizes the quadratic performance index function (Cost) J_M which is defined in (5).

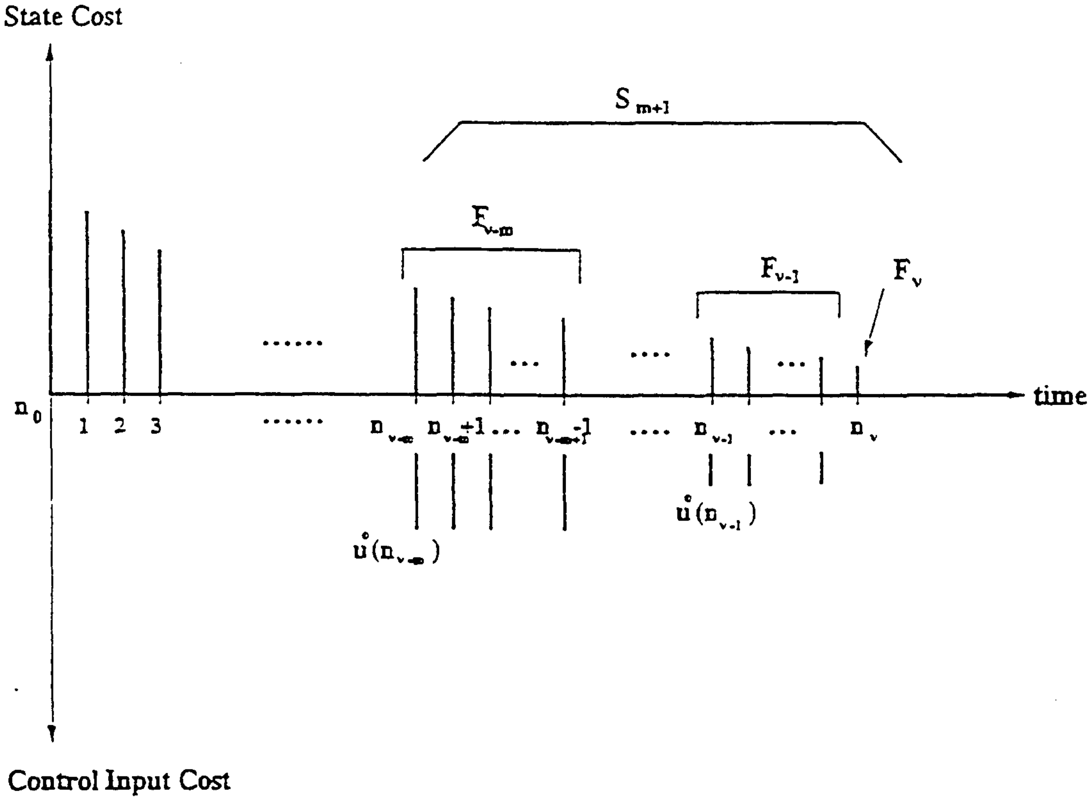


Figure 1: Decomposition of J_M into F_i.

The principle of optimality, developed by Richard Bellman[2, 3] is the approach used here. That is, if a closed loop control $u^o(n_i) = g[x(n_i)]$ is optimal over the interval $t_0 \leq t \leq t_\nu$, then it is also optimal over any sub-interval $t_m \leq t \leq t_\nu$, where $0 \leq m \leq \nu$. As it can be seen from Figure 1, the total cost J_M can be decomposed into F_is for $0 \leq i \leq \nu$ where

$$\begin{aligned} F_i &= x^T(n_i)Qx(n_i) + x^T(n_i+1)Qx(n_i+1) \\ &+ x^T(n_i+2)Qx(n_i+2) + \ldots + x^T(n_{i+1}-1)Qx(n_{i+1}-1) \\ &+ (n_{i+1}-n_i)u^T(n_i)Ru(n_i) \end{aligned} \tag{7}$$

That is, from (1),

$$\begin{aligned} F_i &= x^T(n_i)Qx(n_i) + (Ax(n_i) + Bu(n_i))^T Q(Ax(n_i) + Bu(n_i)) \\ &+ (A^2x(n_i) + ABu(n_i) + Bu(n_i))^T Q(A^2x(n_i) + ABu(n_i) + Bu(n_i)) \\ &+ \ldots + (A^{n_{i+1}-n_i-1}x(n_i) + A^{n_{i+1}-n_i-2}Bu(n_i) + \ldots + ABu(n_i) + Bu(n_i))^T Q \\ &\quad (A^{n_{i+1}-n_i-1}x(n_i) + A^{n_{i+1}-n_i-2}Bu(n_i) + \ldots + ABu(n_i) + Bu(n_i)) \\ &+ (n_{i+1}-n_i)u^T(n_i)Ru(n_i) \end{aligned} \tag{8}$$

This can be rewritten as

$$\begin{aligned} F_i &= x^T(n_i)Qx(n_i) + \sum_{j=1}^{n_{i+1}-n_i-1} [A_jx(n_i) + B_ju(n_i)]^T Q[A_jx(n_i) + B_ju(n_i)] \\ &+ (n_{i+1}-n_i)u^T(n_i)Ru(n_i) \end{aligned} \tag{9}$$

where $A_j = A^j$ and $B_j = \sum_{k=0}^{j-1} A^k B$.

Then J_M can be expressed as

$$J_M = F_0 + F_1 + F_2 + \ldots + F_\nu. \tag{10}$$

Let S_m be the cost from $i = \nu - m + 1$ to $i = \nu$:

$$S_m = F_{\nu-m+1} + F_{\nu-m+2} + \ldots + F_{\nu-1} + F_\nu; \quad 1 \leq m \leq \nu + 1. \tag{11}$$

These cost terms are well illustrated in the above Figure 1.

Therefore, by applying the principle of optimality, we can first minimize $S_1 = F_\nu$, then choose $F_{\nu-1}$ to minimize $S_2 = F_{\nu-1} + F_\nu = S_1^o + F_{\nu-1}$ where S_1^o is the optimal cost occurred at t_ν. We can continue choosing $F_{\nu-2}$ to minimize $S_3 = F_{\nu-2} + F_{\nu-1} + F_\nu = F_{\nu-2} + S_2^o$ and so on until $S_{\nu+1} = J_M$ is minimized. Note that $S_1 = F_\nu = x^T(n_\nu)Qx(n_\nu)$ is determined only from $x(n_\nu)$ which is independent of any other control inputs.

3.2 Inductive Construction of an Optimal Control Law with D_ν Given

We inductively derive an optimal controller which changes its control at ν time instants $t_0, t_1, \ldots, t_{\nu-1}$. As we showed in the previous section, the inductive procedure goes backwards in time from S_1° to $S_{\nu+1}^\circ$. Since $S_1 = F_\nu = x^T(n_\nu)Qx(n_\nu) + u^T(n_\nu)Ru(n_\nu)$ and $x(n_\nu)$ is independent of $u(n_\nu)$, we can let $u^\circ(n_\nu) = u^\circ(M) = 0$ and $S_1^\circ = x^T(n_\nu)Qx(n_\nu)$ where Q is symmetric and positive semi-definite.

Induction Basis: $S_1^\circ = x^T(n_\nu)Qx(n_\nu)$ where Q is symmetric.

Inductive Assumption: Suppose that

$$S_m^\circ = x^T(n_{\nu-m+1})P(\nu-m+1)x(n_{\nu-m+1})$$

holds for some m where $1 \le m \le \nu$ and $P(\nu - m + 1)$ is symmetric.

We can write S_m° as

$$\begin{aligned} S_m^\circ \;=\; & [A_{(n_{\nu-m+1}-n_{\nu-m})}x(n_{\nu-m}) + B_{(n_{\nu-m+1}-n_{\nu-m})}u(n_{\nu-m})]^T P(\nu-m+1) \\ & [A_{(n_{\nu-m+1}-n_{\nu-m})}x(n_{\nu-m}) + B_{(n_{\nu-m+1}-n_{\nu-m})}u(n_{\nu-m})] \end{aligned} \tag{12}$$

From the definition of S_m and (9),

$$\begin{aligned} S_{m+1} &= S_m^\circ + F_{\nu-m} \\ &= S_m^\circ + x^T(n_{\nu-m})Qx(n_{\nu-m}) \\ &+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})]^T Q[A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})] \\ &+ (n_{\nu-m+1} - n_{\nu-m})u^T(n_{\nu-m})Ru(n_{\nu-m}) \end{aligned} \tag{13}$$

And the above equation becomes

$$\begin{aligned} S_{m+1} \;=\; & [A_{n_{\nu-m+1}-n_{\nu-m}}x(n_{\nu-m}) + B_{n_{\nu-m+1}-n_{\nu-m}}u(n_{\nu-m})]^T P(\nu-m+1) \\ & [A_{n_{\nu-m+1}-n_{\nu-m}}x(n_{\nu-m}) + B_{n_{\nu-m+1}-n_{\nu-m}}u(n_{\nu-m})] \\ + \; & x^T(n_{\nu-m})Qx(n_{\nu-m}) \\ + \; & \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})]^T Q[A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})] \\ + \; & (n_{\nu-m+1} - n_{\nu-m})u^T(n_{\nu-m})Ru(n_{\nu-m}) \end{aligned} \tag{14}$$

If we differentiate S_{m+1} with respect to $u(n_{\nu-m})$, then

$$\begin{aligned}
\frac{\partial S_{m+1}}{\partial u(n_{\nu-m})} &= B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) A_{n_{\nu-m+1}-n_{\nu-m}} x(n_{\nu-m}) && (15)\\
&+ (A^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}})^T x(n_{\nu-m})\\
&+ 2B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}} u(n_{\nu-m})\\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [2B_j^T Q A_j x(n_{\nu-m}) + 2B_j^T Q B_j u(n_{\nu-m})]\\
&+ 2(n_{\nu-m+1}-n_{\nu-m}) R u(n_{\nu-m})\\
&= 2\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) A_{n_{\nu-m+1}-n_{\nu-m}} && (16)\\
&\quad + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q A_j\} x(n_{\nu-m})\\
&+ 2\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}}\\
&\quad + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q B_j + (n_{\nu-m+1}-n_{\nu-m}) R\} u(n_{\nu-m})
\end{aligned}$$

Note that $P(\nu-m+1)$ is symmetric and the following three rules are applied to differentiate S_{m+1} above.

$$\begin{aligned}
\frac{\partial}{\partial x}(x^T Q x) &= 2Qx\\
\frac{\partial}{\partial x}(x^T Q y) &= Qy\\
\frac{\partial}{\partial y}(x^T Q y) &= Q^T x
\end{aligned}$$

Let $\frac{\partial S_{m+1}}{\partial u(n_{\nu-m})} = 0$, from Lemma 1 and Lemma 2 given later we can obtain $u^o(n_{\nu-m})$ which minimizes S_{m+1} and thus obtain S^o_{m+1}.

$$\begin{aligned}
u^o(n_{\nu-m}) &= -\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}} && (17)\\
&\quad + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q B_j + (n_{\nu-m+1}-n_{\nu-m}) R\}^{-1}\\
&\quad \{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) A_{n_{\nu-m+1}-n_{\nu-m}} + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q A_j\} x(n_{\nu-m})\\
&= -K(\nu-m) x(n_{\nu-m})
\end{aligned}$$

where $K(\nu-m)$ is defined in (17).

Therefore, we can write

$$\begin{aligned} A_{n_{\nu-m+1}-n_{\nu-m}}x(n_{\nu-m}) \; + \; & B_{n_{\nu-m+1}-n_{\nu-m}}u^{o}(n_{\nu-m}) = \\ & [A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]x(n_{\nu-m}) \end{aligned} \tag{18}$$

If we use (17) and (18), we have

$$\begin{aligned} S^{o}_{m+1} \;=\; & \{[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]x(n_{\nu-m})\}^{T}P(\nu-m+1) \\ & \{[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]x(n_{\nu-m})\} \\ + \; & x^{T}(n_{\nu-m})Qx(n_{\nu-m}) \\ + \; & \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} \{[A_j - B_jK(\nu-m)]x(n_{\nu-m})\}^{T}Q\{[A_j - B_jK(\nu-m)]x(n_{\nu-m})\} \\ + \; & (n_{\nu-m+1} - n_{\nu-m})[K(\nu-m)x(n_{\nu-m})]^{T}R[K(\nu-m)x(n_{\nu-m})] \end{aligned} \tag{19}$$

This equation can be rewritten as

$$\begin{aligned} S^{o}_{m+1} \;=\; & x^{T}(n_{\nu-m})\{[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]^{T}P(\nu-m+1) \\ & [A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)] \\ + \; & Q \\ + \; & \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_j - B_jK(\nu-m)]^{T}Q[A_j - B_jK(\nu-m)] \\ + \; & (n_{\nu-m+1} - n_{\nu-m})K^{T}(n_{\nu-m})RK(\nu-m)\}x(n_{\nu-m}). \\ = \; & x^{T}(n_{\nu-m})P(\nu-m)x(n_{\nu-m}) \end{aligned} \tag{20}$$

where $P(\nu-m)$ is obtained from $K(\nu-m)$ and $P(\nu-m+1)$ as in (20). Also note that knowing $P(\nu-m+1)$ is enough to compute $K(\nu-m)$ because other terms of (17) are known a priori.

Therefore, we find a symmetric matrix $P(\nu-m)$ satisfying $S^{o}_{m+1} = x^{T}(n_{\nu-m})P(\nu-m)x(n_{\nu-m})$. From (17) and (20), we have the following recursive equations for obtaining $P(\nu-m)$ from $P(\nu-m+1)$ where $m = 1, 2, ..., \nu$.

$$\begin{aligned} K(\nu-m) \;=\; & \{B^{T}_{n_{\nu-m+1}-n_{\nu-m}}P(\nu-m+1)B_{n_{\nu-m+1}-n_{\nu-m}} \\ + \; & \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^{T}QB_j + (n_{\nu-m+1} - n_{\nu-m})R\}^{-1} \\ & \{B^{T}_{n_{\nu-m+1}-n_{\nu-m}}P(\nu-m+1)A_{n_{\nu-m+1}-n_{\nu-m}} + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^{T}QA_j\} \end{aligned} \tag{21}$$

$$\begin{aligned} P(\nu - m) &= [A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}} K(\nu - m)]^T P(\nu - m + 1) \\ & \quad [A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}} K(\nu - m)] \\ &+ Q \\ &+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_j - B_j K(\nu - m)]^T Q [A_j - B_j K(\nu - m)] \\ &+ (n_{\nu-m+1} - n_{\nu-m}) K^T(\nu - m) R K(\nu - m) \end{aligned} \tag{22}$$

Also, we know that at each time instant $n_{\nu-m}\Delta$

$$u^o(n_{\nu-m}) = -K(\nu - m) x(n_{\nu-m}) \tag{23}$$

Hence, with $P(\nu) = Q$, we can obtain $K(i)$ and $P(i)$ for $i = \nu - 1, \nu - 2, ..., 0$ recursively using (21) and (22). At each time instant $n_i\Delta$, $i = 0, 1, 2, ..., \nu - 1$ the new control input value will be obtained using (23) by multiplying $K(i)$ by $x(n_i)$ where $x(n_i)$ is the estimate of the system state at $n_i\Delta$. Also, note that the optimal control cost is $J_M^o = S_{\nu+1}^o = x^T(0)P(0)x(0)$ where $P(0)$ is found from the above procedure.

To prove the optimality of this control law we need the following lemmas.

Lemma 1 *If Q is positive semi-definite and R is positive definite, then $P(i)$, $i = \nu, \nu-1, \nu-2, ..., 0$, matrices are positive semi-definite. Hence, $P(i)$s are symmetric from the definition of a positive semi-definite matrix.*

Proof Since $P(\nu) = Q$, from assumption $P(\nu)$ is positive semi-definite. Assume that for $k = i + 1$, $P(k)$ is positive semi-definite. We use induction to prove that $P(i)$ is semi-definite. Note that Q is positive semi-definite and R is positive definite. From (22) we have

$$\begin{aligned} P(i) &= [A_{n_{i+1}-n_i} - B_{n_{i+1}-n_i} K(i)]^T P(i + 1) \\ & \quad [A_{n_{i+1}-n_i} - B_{n_{i+1}-n_i} K(i)] \\ &+ Q \\ &+ \sum_{j=1}^{n_{i+1}-n_i-1} [A_j - B_j K(i)]^T Q [A_j - B_j K(i)] \\ &+ (n_{i+1} - n_i) K^T(i) R K(i) \end{aligned} \tag{24}$$

Since $P(i+1)$ and Q are positive semi-definite, R is positive definite, and $(n_{i+1} - n_i) > 0$, it is easy to verify that for $\forall y \in R^m \ : \ y^T P(i) y \geq 0$. This means that $P(i)$ is positive semi-definite. This inductive procedure proves the lemma.

Lemma 2 *Given D_ν, the inverse matrix in (21) always exists.*

Proof Let $V = B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu - m + 1) B_{n_{\nu-m+1}-n_{\nu-m}} + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q B_j + (n_{\nu-m+1} - n_{\nu-m})R$. From Lemma 1, $P(\nu - m + 1)$ is positive semi-definite. Therefore, $\forall y \in R^m$: $y^T V y > 0$ because Q is positive semi-definite, R is positive definite and $n_{\nu-m+1} - n_{\nu-m} > 0$. This implies that V is positive definite. Hence the inverse matrix exists.

Theorem 1 *Given D_ν, $K(i)$ $(i = 0, 1, 2, ..., \nu - 1)$ obtained from the above procedure are the optimal feedback gains which minimize the cost function J_M (and J'_M) on* $[0, M\Delta]$.

Proof Note that given D_ν, J_M is a convex function of $u(n_i), i = 0, 1, ..., \nu - 1$. Thus the above feedback control law is optimal.

Lemma 3 *If $p \leq q$ and $D_p \subseteq D_q$, then $J^o_{M_p} \geq J^o_{M_q}$ where $J^o_{M_p}$ and $J^o_{M_q}$ are the optimal costs of controls which change controls at time instants in D_p and D_q respectively.*

Proof Suppose that $J^o_{M_p} < J^o_{M_q}$, then, in controlling the system with D_q, if we do not change controls at time instants in $D_q - D_p$ and change controls at time instants in D_p to the same control inputs that were exercised to get $J^o_{M_p}$ with D_p, we obtain $\hat{J}_{M_q}$ which is equal to $J^o_{M_p}$. This contradicts the fact that $J^o_{M_q}$ is the minimum cost obtainable with D_q since we have found $\hat{J}_{M_q}$ which is equal to $J^o_{M_p}$ and therefore less than $J^o_{M_q}$. Hence, $J^o_{M_p} \geq J^o_{M_q}$.

This lemma implies that if we do not take computation cost, μ, into consideration, then the more control exercising points, the better the controller is (less cost). With the computation cost being included in the cost function, the statement above is no longer true. Therefore we need to search for an optimal D_ν which minimizes the cost function J'_M. The following sections provide a detailed discussion on searching for such an optimal solution. Note that if we let $D_\nu = D_M$ then the optimal temporal control law is the same as the traditional linear feedback optimal control law.

3.3 Optimal Temporal Control Law over D_ν Space with ν Given

When the number of control changing points, ν, and an initial system state $x(0)$ are given, we search over a set of possible D_νs and $u(D_\nu)$s such that the cost function J_M is minimized. This can be done by varying $\nu - 1$ control changing time instants, t_i, $i = 1, 2, ..., \nu - 1$ (since $t_0 = 0$) over the discrete set, $D_M = \{0, \Delta, 2\Delta, \ldots, (M-1)\Delta\}$ and applying the technique developed in the previous section for each given D_ν. Let us denote such a D_ν which minimizes J_M as D^o_ν. Note that when ν is given, minimizing J_M is equivalent to minimizing J'_M. Since both D_ν and $u(D_\nu)$ are control variates, to be able to find a global optimal solution, either an exhaustive search or some global search methods like *Genetic Algorithm* or *Simulated Annealing* should be considered. Later we present a numerical example in which an exhaustive search with *Steepest Descent Search* method is used. Searching for a globally optimal solution for a temporal controller calls for further research.

3.4 Optimal Temporal Control Law

Assume that a maximum number of control changing points, ν_{max}, is given. By varying ν from 1 to ν_{max} we can find D^o_ν. to obtain a globally optimal temporal controller which minimizes J'_M. This can be done by first searching for D^o_ν for each given ν and then comparing the cost function $J'_M = J_M + \nu\mu$ at each D^o_ν, $\nu = 1, 2, \ldots, \nu_{max}$. That is, let $J'^o_{M_\nu} = x^T(0)P(0)x(0) + \nu\mu$ where $P(0)$ is calculated at D^o_ν as in the previous section. Then we can obtain a global minimum cost $J'^o_M = \min_{1 \le \nu \le \nu_{max}} \{J'^o_{M_\nu}\}$ and an optimal number of control changes, ν^o, at which $J'^o_{M_{\nu^o}} = J'^o_M$.

3.5 Terminal State Constraints

The terminal state constraints may be used to check if the optimal temporal controller with D^o_ν. can drive the system state to a permissible final state within a given time. Let X_f be a set of allowed terminal states, if $x(n_\nu) \in X_f$, then the control law is said to be *stable* in terms of the terminal state constraints and *not stable* if $x(n_\nu) \notin X_f$. If the globally optimal temporal controller obtained from the above procedure is not stable, ν^* should be increased until a stable one is found. One way of specifying terminal state constraints for regulators might be $| x(M)_i | \le \epsilon_i$ where $x(M)_i$ is the ith element of $x(M)$ state vector.

3.6 Algorithm to Derive an Optimal Temporal Controller

To summarize the above discussion, we provide in Figure 2 a complete algorithm to search for a globally optimal temporal controller under the assumption that the initial state $x(0)$ is given.

In the algorithm, a neighbor of $D_\nu = \{n_0\Delta, n_1\Delta, n_2\Delta, \ldots, n_{\nu-1}\Delta\}$ is defined to be any member of a set $N(D_\nu) = \{\{n_0'\Delta, n_1'\Delta, \ldots, n_{\nu-1}'\Delta\} \mid \mid n_i' - n_i \mid \leq 1,\ i = 1, 2, \ldots, \nu - 1\}$.

3.7 Optimal Temporal Controllers over an Initial State Space

Note that D_ν^o might become different if a new initial system state $\hat{x}(0)$ is used instead of $x(0)$ when the state vector is in $R^{m\times 1}$ where $m \geq 2$. This is because the cost function $J_M = x^T(0)P(0)x(0)$ depends on $x(0)$ as well as $P(0)$. Thus, D_ν^o is dependent on the initial state $x(0)$. However, when $m = 1$ it can be shown that D_ν^o is independent of any initial state. To see this let $x(0) = k\hat{x}(0) \in \mathcal{R}^1$ and $P(0)$ and $\hat{P}(0)$ be the optimal matrices with initial states $x(0)$ and $\hat{x}(0)$, respectively. i.e.,

$$J_M(x(0)) = x(0)P(0)x(0)$$
$$J_M(\hat{x}(0)) = \hat{x}(0)\hat{P}(0)\hat{x}(0)$$

From the optimality of $\hat{P}(0)$ with respect to $\hat{x}(0)$,

$$\hat{x}^T(0)P(0)\hat{x}(0) \;\geq\; \hat{x}^T(0)\hat{P}(0)\hat{x}(0) \tag{25}$$

Multiplying the above inequality by k^2 we have

$$\begin{aligned} k^2\hat{x}^T(0)P(0)\hat{x}(0) &= x^T(0)P(0)x(0) \\ &\geq k^2\hat{x}^T(0)\hat{P}(0)\hat{x}(0) \\ &= x^T(0)\hat{P}(0)x(0) \end{aligned} \tag{26}$$

On the other hand, due to the optimality of $P(0)$ we have

$$x^T(0)\hat{P}(0)x(0) \;\geq\; x^T(0)P(0)x(0) \tag{27}$$

Therefore, $\hat{P}(0) = P(0)$. This implies the optimality of $\hat{P}(0)$ and $\hat{D}_\nu^o$ for any initial state $x(0) \in \mathcal{R}^1$.

Generally speaking, the above result will not hold for $m \geq 2$ cases. However, using the same argument discussed above we can prove that for any initial state $x(0) = k\hat{x}(0)$, $x(0)$ and $\hat{x}(0)$ will have the same D_ν^o as well as the same $P(0)$.

```
$\nu^o = 1$
$J'^{o}_{M} = \infty$
for $\nu = 1$ to $\nu_{max}$ {
    /* Several different search starting points */
    for $i = 1$ to $NumInitPts_\nu$ {
        $D_\nu = D^{init,i}_\nu$
        /* Iterate until a local minimum is found - Steepest Descent Search */
        while (MinimumFound != True) {
            Find optimal costs for neighboring points of $D_\nu$ using theorem 1
            if ( $J'_M$ has a Local Minimum at $D_\nu$)
                then {
                        MinimumFound = True
                        $J'^{i}_{M_\nu}$ = Cost($J'_M$) at $D_\nu$ }
                else
                        $D_\nu$ = a neighbor of $D_\nu$ with the smallest $J'_M$
            }
        }
    $J'^{o}_{M_\nu} = \min_{1 \le i \le NumInitPts_\nu}\{J'^{i}_{M_\nu}\}$
    if ( $J'^{o}_{M_\nu} < J'^{o}_{M}$ )
        then {
                $\nu^o = \nu$
                $J'^{o}_{M} = J'^{o}_{M_\nu}$ }
    }
```

Figure 2: Complete algorithm to find an optimal temporal controller.

4 Implementation

To implement temporal control, we need to calculate and store $K(i)$ matrices in (22) and use them when controlling the system utilizing (23). Note that in traditional optimal linear control a similar matrix is obtained and used at every time instant in D_M to generate control input value. While the feedback gain matrices for traditional linear optimal controller are independent of initial states, the number of control exercises, ν, and $K(i)$ matrices are dependent on initial states for temporal control systems. But, if the possible set of initial states is in $\mathcal{R}^1$ they are independent of the initial states. Effective deployment of temporal control requires that we know the range of initial state values and generate $K(i)$ matrices for each group. A sensitivity analysis is required to determine how many distinct matrices need to be stored.

In order to implement temporal control we require an operating system that supports scheduling control computations at specific time instants. The Maruti system developed at the University of Maryland is a suitable host for the implementation of temporal control [10, 8, 7]. In Maruti, all executions are scheduled in time and the time of execution can be modified dynamically, if so desired. This is in contrast with traditional cyclic executives often used in real-time systems, which have a fixed, cyclic operation and which are well suited only for the sampled data control systems operating in a static environment. It is the availability of the system such as Maruti that allows us to consider the notion of temporal control, in which time becomes an emergent property of the system.

5 Example

To illustrate the advantages of a temporal control scheme let us consider a simple example of rigid body satellite control problem [12]. The system state equations are as follows:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0.00125 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} 1 & 1 \end{bmatrix} x(k) \end{aligned}$$

where k represents the time index and one unit of time is the discretized subinterval of length $\Delta = 0.05$. The linear quadratic performance index J'_M in (5) is used here with the following parameters.

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

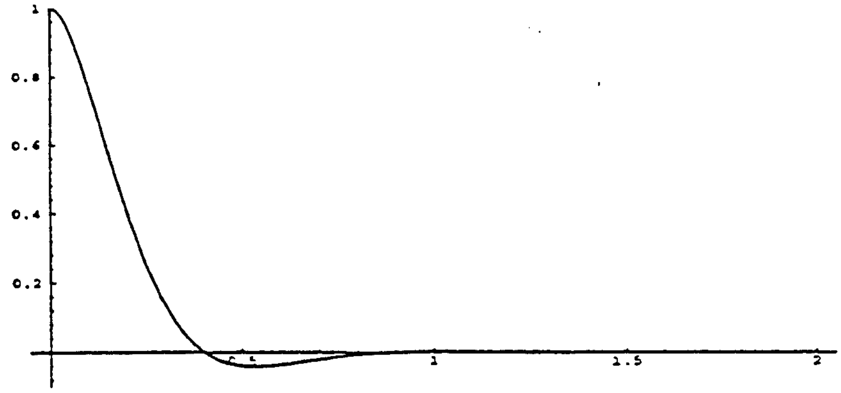


Figure 3: Optimal Linear Control with $\Delta = 0.05$.

$$\begin{aligned} R &= 0.0001 \\ \mu &= 0.02 \ \& \ 0.01 \\ M &= 40 \\ \Delta &= 0.05 \\ \epsilon_i &= 0.01, \quad i = 1, 2 \\ x(0) &= \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \end{aligned} \tag{28}$$

The objective of the control is to drive the satellite to the zero position and the desired goal state is $x_f = [0, \ \ 0]^T$. The terminal state constraint is $| \ x_i(40) \ | \leq \epsilon_i \quad i = 1, 2$. With the equal sampling interval $\Delta = 0.05$ and $M = 40$ the optimal linear feedback control of this system has cost function $J_M = 0.984678$ (without computational cost) and $J_M' = 1.784678$ (with computational cost) and is shown in Figure 3. The terminal state constraint is satisfied at $0.8sec$.

If we apply the temporal control scheme presented above to this problem with $\mu = 0.02$ we find that the optimal number of control changes for this example is 3 and $D_3^o = \{0, 2\Delta, 10\Delta\}$ with a cost $J_M' = 1.08388$. Note that the 40 step optimal linear feedback controller given above has a cost $J_M' = 1.784678$ when computation cost is considered. Table 1 shows how this optimal controller is obtained when we set $\nu_{max} = 7$. Figure 4(a) shows the system trajectory when this three-step optimal temporal controller is used to control the system. This trajectory satisfies the terminal state constraint at $0.8sec$ as well. Also, the maximum control input magnitudes, $| \ u \ |_{max}$, in both

n	D^o_ν	Cost(J'_M) with $\mu = 0.02$	Cost(J'_M) with $\mu = 0.01$
1	$\{0\}$	$4.63089 + \mu = 4.65089$	$4.63089 + \mu = 4.64089$
2	$\{0,1\}$	$1.44603 + 2\mu = 1.48603$	$1.44603 + 2\mu = 1.46603$
3	$\{0,2,10\}$	$1.02388 + 3\mu = 1.08388$	$1.02388 + 3\mu = 1.05388$
4	$\{0,2,9,11\}$	$1.02224 + 4\mu = 1.10224$	$1.02224 + 4\mu = 1.06224$
5	$\{0,1,3,8,11\}$	$0.996968 + 5\mu = 1.096968$	$0.996968 + 5\mu = 1.046968$
6	$\{0,1,3,8,11,24\}$	$0.996746 + 6\mu = 1.116746$	$0.996746 + 6\mu = 1.056746$
7	$\{0,1,3,8,11,23,25\}$	$0.996745 + 7\mu = 1.136745$	$0.996745 + 7\mu = 1.066745$

Table 1: Calculating optimal temporal controllers.

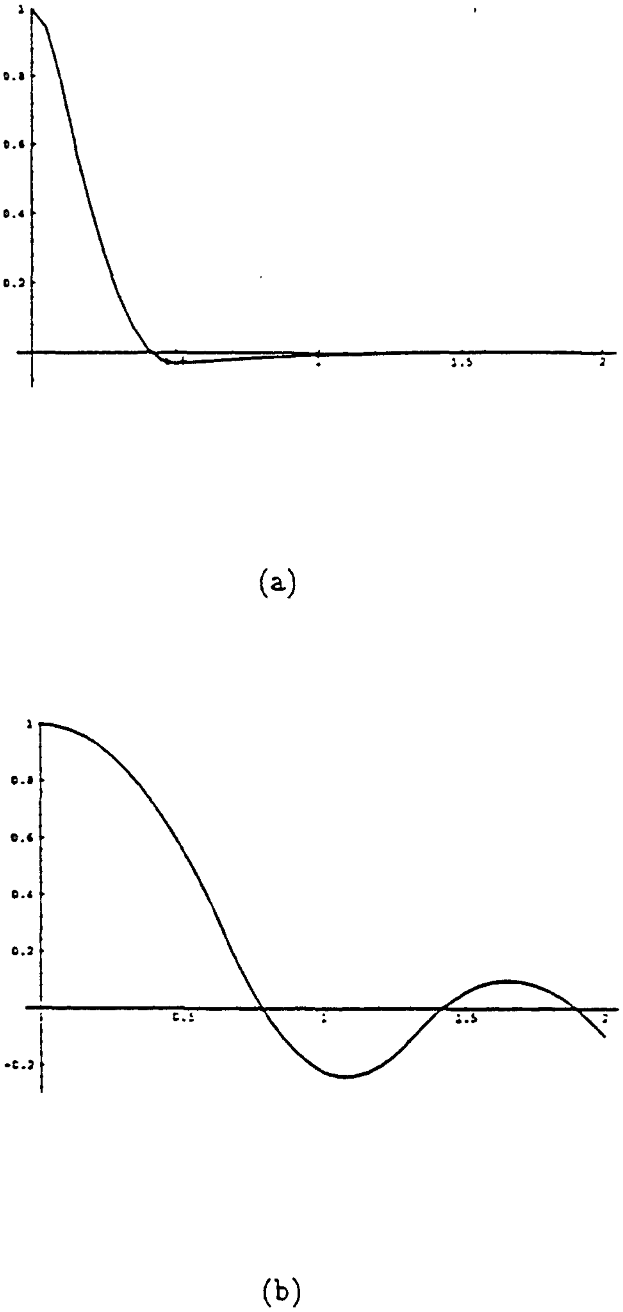
controllers lie within the same bound $B = 50$, which may be another constraint on control.

The optimal temporal controller found with $\mu = 0.01$ has $\nu = 5$ and $D^o_5 = \{0, \Delta, 3\Delta, 8\Delta, 11\Delta\}$ with a cost $J_M = 0.996968$. Note that this cost is even less than 1.01269 which is obtained from the optimal controller with equal sampling period $0.1sec$ and 20 control changes.

If we change control values only at three time instants with equal sampling period, $13M = 0.65sec$, the total cost incurred is 2.2823(without computational cost) on the time interval $[0, 2]$. The cost is more than twice that of our optimal temporal controller and the terminal state constraint is not satisfied even at the end of the controlling interval of $2.0sec$. Figure 4(b) clearly shows the advantages of using an optimal temporal controller over using an optimal controller of equidistant samplings. Their performances are noticeably different though both of them are changing controls at three time instants. It is clear that the optimal temporal control with three control changes performs almost the same as 40 step linear optimal controller does. This implies that enforcing the constant sampling rate throughout the entire controlling interval may simply waste computational power which otherwise could be used for other concurrent controlling tasks in critical systems.

Obtaining D^o_3 for this example was simple since J_{40} has only one minimum over the entire set of possible D_3s on $[0, 40\Delta]$. Figure 5(a) and Figure 5(b) show that J_{40} has only one local(global) minimum at $D^o_3 = \{0, 2\Delta, 10\Delta\}$. We got this optimal D_3 by doing steepest descent search with the starting point $D^{init}_3 = \{0, \Delta, 10\Delta\}$ after searching for only three points, $\{0, \Delta, 10\Delta\}$, $\{0, 2\Delta, 10\Delta\}$, $\{0, 3\Delta, 10\Delta\}$. Also, Figure 5(a) shows that choosing n_1 has greater influence on the total cost than n_2 since the cost varies more radically along the n_1 axis in the figure. This means that the initial stage of the control needs more attention than the later stage in this linear control problem.

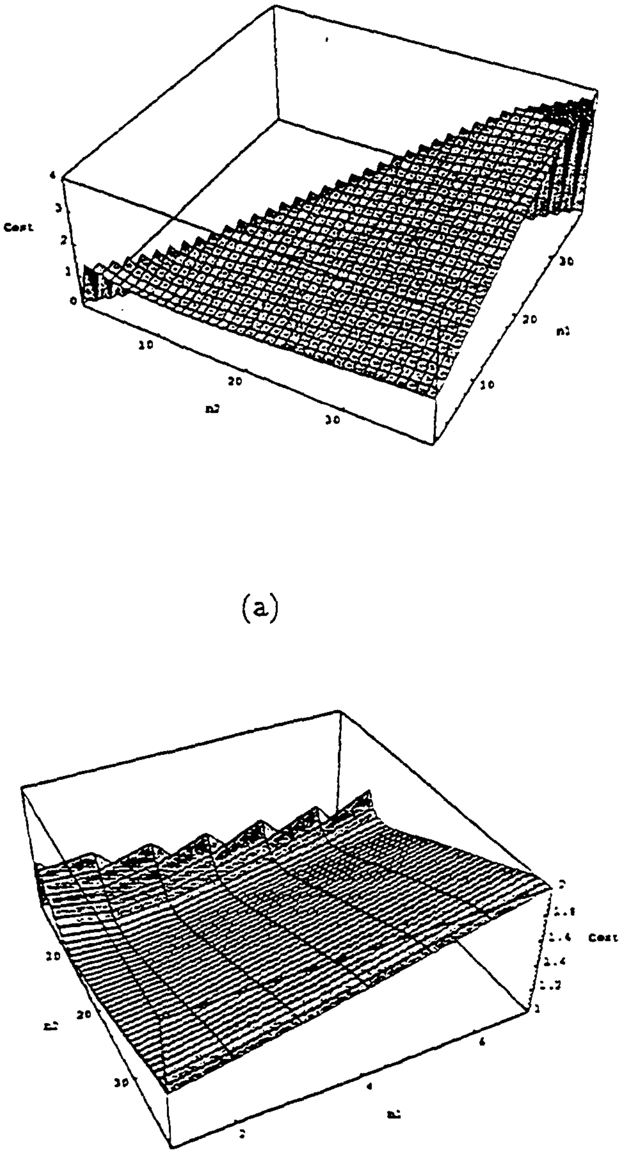
But, if we change one of the parameters of performance index function, R, from 0.0001 to 0.001 we get two local minima at $D^1_3 = \{0, \Delta, 2\Delta\}$ and $D^2_3 = \{0, 3\Delta, 19\Delta\}$, among which D^2_3 is the



(a)

(b)

Figure 4: Control trajectories with 3 control changes. (a)Optimal temporal control with $D_3^o = \{0, 2\Delta, 10\Delta\}$. (b)Optimal linear control with 13Δ (0.65*sec*) period.



(a)

(b)

Figure 5: Cost function distribution over (n_1, n_2). (a)Costs on D_3 space. (b)Costs near $D_3^o = \{0, 2\Delta, 10\Delta\}$.

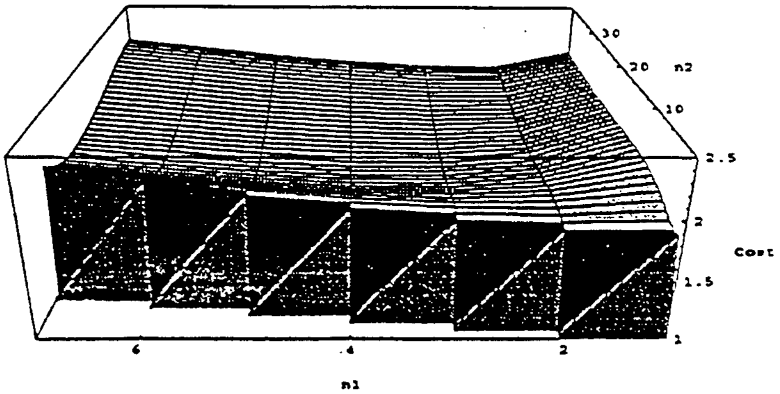


Figure 6: Costs near D_3^1 and D_3^2 with $R = 0.001$.

optimal one with less cost. Figure 6 shows this fact. In this case we need to use steepest descent search method at least twice with different search starting points to get an optimal solution. We implemented this steepest descent search algorithm in Mathematica and used it to generate D_ν^o for several examples by varying ν. For our examples of linear time invariant system control problems the number of local minima was not so large that we could efficiently apply this search method just a few times with different initial D_ν^{init}s to get a global minimum without doing an exhaustive search over the entire D_ν space.

6 Discussion

Employing the temporal control methodology in concurrent real-time embedded systems will have a significant impact on the way computational resources are utilized by control tasks. A minimal amount of control computations can be obtained for a given regulator by which we can achieve almost the same control performance compared to that of traditional controller with equal sampling period. This significantly reduces the CPU times for each controlling task and thus increases the number of real-time control functions which can be accommodated concurrently in one embedded system. Particularly, in a hierarchical control system if temporal controllers can be employed for lower level controllers the higher level controllers will have a great degree of flexibility in managing resource usages by adjusting computational requirements of each lower level controller. For example, in emergency situations the higher level controller may force the lower level controller to run as infrequently as they possibly can (thus freeing computational resources for handling the emergency). In contrast, during normal operations the temporal control tasks may run as necessary, and the additional computation time can be used for higher level functions such as monitoring and planning, etc.

In addition, the method developed in Section 3.2, which calculates an optimal controller when control changing time instants are given, can be applied to the case in which the control computing time instants cannot be periodic. For example, when a small embedded controller is used to control several functions, it may be a lot better to design a temporal controller for each function such that the required computational resources are appropriately scheduled while retaining the required degree of control for each function.

7 Conclusion

In this paper we proposed a *temporal control* technique based on a new cost function which takes into account computational cost as well as state and input cost. In this scheme new control input values are defined at time instants which are not necessarily regularly spaced. For the linear control problem we showed that almost the same quality of control can be achieved while much less computations are used than in a traditional controller.

The proposed formulation of temporal control is likely to have a significant impact on the way concurrent embedded real-time systems are designed. In hierarchical control environment, this approach is likely to result in designs which are significantly more efficient and flexible than traditional control schemes. As it uses less computational resources, the lower level temporal controllers will make the resources available to the higher level controllers without compromising the quality of control.

References

[1] A. Belleisle. Stability of systems with nonlinear feedback through randomly time-varying delays. *IEEE Transactions on Automatic Control*, AC-20:67–75, February 1975.

[2] R. Bellman. *Adaptive Control Process: A Guided Tour.* Princeton,NJ: Princeton University Press, 1961.

[3] R. Bellman. Bellman special issue. *IEEE Transactions on Automatic Control*, AC-26, October 1981.

[4] P. Dorato and A. Levis. Optimal linear regulators: The discrete time case. *IEEE Transactions on Automatic Control*, AC-16:613–620, December 1971.

[5] A. Gosiewski and A. Olbrot. The effect of feedback delays on the performance of multivariable linear control systems. *IEEE Transactions on Automatic Control*, AC-25(4):729–734, August 1980.

[6] K. Hirai and Y. Satoh. Stability of a system with variable time delay. *IEEE Transactions on Automatic Control*, AC-25(3):552–554, June 1980.

[7] S. T. Levi, Satish K. Tripathi, Scott Carson, and Ashok K. Agrawala. The MARUTI hard real-time operating system. *ACM Symp. on Op. Syst. Principles, Op. Syst. Review*, 23(3), July 1989.

[8] Shem-Tov Levi and Ashok K. Agrawala. *Real Time System Design*. McGraw Hill, 1990.

[9] Z. Rekasius. Stability of digital control with computer interruptions. *IEEE Transactions on Automatic Control*, AC-31:356–359, April 1986.

[10] Manas Saksena, James da Silva, and Ashok K. Agrawala. *Design and Implementation of Maruti-II*, chapter 4. Prentice Hall, 1995. In *Advances in Real-Time Systems*, edited by Sang H. Son.

[11] K. G. Shin and H. Kim. Derivation and application of hard deadlines for real-time control systems. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1403–1413, November 1992.

[12] G.S. Virk. *Digital Computer Control Systems*, chapter 4. McGraw Hill, 1991.

[13] K. Zahr and C. Slivinsky. Delay in multivariable computer controlled linear systems. *IEEE Transactions on Automatic Control*, pages 442–443, August 1974.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
July 1995

3. REPORT TYPE AND DATES COVERED
Technical Report

4. TITLE AND SUBTITLE
Designing Temporal Controls

5. FUNDING NUMBERS
N00014-91-C-0195 and
DSAG-60-92-C-0055

6. AUTHOR(S)
Ashok K. Agrawala, Seonho Choi & Leyuan Shi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Maryland
A.V. Williams Building
College Park, Maryland 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
CS-TR-3504
UMIACS-TR-95-81

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Honeywell
3660 Technology Drive
Minneapolis, MN 55418

Phillips Labs
3550 Aberdeen Ave. SE
Kirtland AFB, NM
87117-5776

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Traditional control systems have been designed to exercise control at regularly spaced time instants. When a discrete version of the system dynamics is used, a constant sampling interval is assumed and a new control value is calculated and exercised at each time instant. In this paper we formulate a new control scheme, temporal control, in which we not only calculate the control value but also decide time instants when the new values are to be used. Taking a discrete, linear, time-invariant system, and a cost function which reflects a cost for computation of the control values, as an example, we show the feasibility of using this scheme. We formulate the temporal control scheme as a feedback scheme and, through a numerical example, demonstrate the significant reduction in cost through the use of temporal control.

14. SUBJECT TERMS
Computing Methodologies

15. NUMBER OF PAGES
22 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRAC
Unclassified	Unclassified	Unclassified	Unlimited

Scheduling an Overloaded Real-Time Sysτem [*]

Shyh-In Hwang, Chia-Mei Chen, and Ashok K. Agrawala

Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland, College Park, MD 20742

Abstract

The real-time systems differ from the conventional systems in that every task in the real-time system has a timing constraint. Failure to execute the tasks under the timing constraints may result in fatal errors. Sometimes, it may be impossible to execute all the tasks in the task set under their timing constraints. Considering a system with limited resources, one solution to handle the overload problem is to reject some of the tasks in order to generate a feasible schedule for the rest. In this paper, we consider the problem of scheduling a set of tasks without preemption in which each task is assigned criticality and weight. The goal is to generate an optimal schedule such that all of the critical tasks are scheduled and then the non-critical tasks are included so that the weight of rejected non-critical tasks is minimized. We consider the problem of finding the optimal schedule in two steps. First, we select a permutation sequence of the task set. Secondly, a pseudo-polynomial algorithm is proposed to generate an optimal schedule for the permutation sequence. If the global optimal is desired, all permutation sequences have to be considered. Instead, we propose to incorporate the simulated annealing technique to deal with the large search space. Our experimental results show that our algorithm is able to generate near optimal schedules for the task sets in most cases while considering only a limited number of permutations.

[*] This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Honeywell or Army/Phillips.

1 Introduction

Real-time computer systems are essential for all embedded applications, such as robot control, flight control, and medical instrumentation. In such systems, the computer is required to support the execution of applications in which the timing constraints of the tasks are specified by the physical system being controlled. The correctness of the system depends on the temporal correctness as well as the functional correctness of the tasks. Failure to satisfy the timing constraints can incur fatal errors. How to schedule the tasks so that their timing constraints are met is crucial to the proper operation of a real-time system.

As an example of an embedded system, let us consider the air defense system which monitors an air space continuously using radars. Whenever an intruder is identified, the embedded control system characterizes it and proceeds to initiate the responsive action in a timely manner. The temporal constraints for this phase of processing are different depending on the intruder, whether it is a missile, a fighter, a bomber, a dummy, etc. Such a system is designed to handle a number of intruders concurrently. If the processing requests exceed the capacity of the system, we expect the system to handle a set of the most significant intruders, and not any arbitrary set of intruders. This involves rejecting the processing of some real-time tasks based on their importance. In this paper, we consider the problem of creating a schedule for a set of tasks such that all critical tasks are scheduled, and then, among the non-critical tasks we select those which can be scheduled feasibly while maximizing the sum of the weights of selected non-critical tasks.

As all systems have finite resources, their ability to execute a set of tasks while meeting the temporal requirements is limited. Clearly, overload conditions may arise if more tasks have to be processed than the available set of resources can handle. Under such overload conditions, we have two choices. We may augment the resources available, or reject some tasks (or both). In [8], a technique was presented to handle transient overloads by taking advantage of redundant computing resources. Another permissible solution to this problem is to reject some of the tasks in order to generate a feasible schedule for the rest. Once a task is accepted by the system, the system should be able to finish it under its timing constraint. Some algorithms may have been shown to perform well under low or moderate resource utilization. However, their performance degrades if the system is overloaded [2]. For example, the EDF algorithm has been shown to be optimal for a periodic task set [6]. If there exists a feasible schedule for the task set, EDF can come up with one. However, if the task set is not feasible, EDF may perform unsatisfactorily. The reason is that a task with urgent deadline may not be able to finish before its deadline. But, due to its urgent deadline, the task has a high priority to use the processor and thus keeps wasting the CPU time until the task expires after its deadline. The waste of CPU time may further prevent other tasks from meeting their deadlines. The other problem is that there is little control over which tasks will meet their deadlines and which will not.

For an overloaded system, how to select tasks for rejection on the basis of their importance becomes a significant issue. When the tasks have equal weight, an optimal schedule can be defined to be one in which the number of rejected tasks is minimized. In our previous study [3], we used a *super sequence* based scheduling algorithm to compute the optimal schedule for the tasks. In this paper, the criticality of the tasks are taken into consideration. Basically, if a task can not meet its deadline, it is rejected so that the CPU time would not be wasted. Secondly, we would like to schedule tasks such that the less important tasks may be rejected in favor of the more important tasks. We classify tasks into two categories: *critical* and *non-critical*. The critical tasks are crucial to the system such that they must not be rejected. The non-critical tasks are given weights to reflect their importance, and are allowed to be rejected. A schedule is *feasible* if all critical tasks in the task set are accepted and are guaranteed to meet their timing constraints. If there exists no feasible schedule for the task set, the task set is considered infeasible. The *loss* of a schedule is defined to be the sum of the weights of the rejected non-critical tasks. A schedule is *optimal* if it is feasible and the loss of the schedule is minimum.

We first propose a Permutation Scheduling Algorithm (PSA) to generate an optimal schedule for a permutation, which is a well defined ordering of tasks. When it comes to scheduling a task set of n tasks, in the worst case there might be up to $n!$ permutations to consider. We propose a Set Scheduling Algorithm (SSA) which incorporates the *simulated annealing* technique [9] to deal with the large search space of permutations. PSA is invoked by SSA to compute the optimal schedule for each permutation. Taking the feedback from the schedulability and loss of the schedule generated by PSA, SSA is able to control the progress of search for an optimal schedule for the task set. Our experimental results show that SSA is able to generate feasible schedules for task sets consisting of 100 tasks with success ratios no less than 98% and loss ratios less than 10% for most cases while searching less than 5,000 permutations. For each permutation, the average number of schedules computed to generate an optimal schedule by PSA, which is invoked by SSA, is usually less than 500. The SSA algorithm can be considered efficient in dealing with the exponential search space for coming up with a satisfactorily near optimal schedule.

In the following section, we define the scheduling problem. In section 3, we present the idea about how to schedule a permutation. In section 4, we incorporate the technique of simulated annealing and discuss how to schedule a task set. In section 5, the results of our experiments are presented, which is followed by our conclusion.

2 The Problem

A task set is represented as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. A task τ_i can be characterized as a record of (r_i, c_i, d_i, w_i), representing the ready time, computation time, deadline, and criticality of the ith task. Time is expressed as a real number. A task can not be started before its ready time. Once started, the task must use the processor without preemption for c_i time units, and be finished by its deadline. If a task is very important for the system such that rejection of the task is not allowed, w_i is set to be CRITICAL. Otherwise, w_i is assigned an integral value to indicate its importance, and is subject to rejection if necessary. A *permutation sequence*, or simply abbreviated to a *permutation*, is an ordered sequence of tasks in the task set. Scheduling is a process of binding starting times to the tasks such that each task executes according to the schedule. Note that a non-preemptive schedule on a single processor implies a sequence for the execution of tasks. For the convenience of our discussion, we hereafter use a sequence to represent the schedule in the context. A permutation is denoted by $\mu = \langle \tau_1, \ldots, \tau_n \rangle$, where τ_i is the ith task in the permutation. A prefix of a permutation is denoted by $\mu_k = \langle \tau_1, \ldots, \tau_k \rangle$.

To schedule a task set, we need to take into consideration the possible permutations in the task set. We first consider an algorithm for scheduling a permutation. The finish time of a schedule is the finish time of the last task in the schedule. Let $S_k(t)$ denote a schedule of μ_k with finish time no more than t. We use $W(S_k(t))$ to represent the weight of $S_k(t)$, which is the sum of the weights of non-critical tasks in the schedule. A feasible schedule of μ_k is defined as follows:

Definition: $S_k(t)$, $1 \leq k \leq n$, is a *feasible* schedule of μ_k at t, if and only if:

1. $S_k(t)$ is a subsequence of μ_k,
2. the finish time of $S_k(t)$ is less than or equal to t, and
3. all critical tasks in μ_k are included in $S_k(t)$.

An optimal schedule of μ_k is defined as follows:

Definition: $\sigma_k(t)$ is an *optimal* schedule of μ_k at t, if and only if:

1. $\sigma_k(t)$ is a feasible schedule of μ_k, and
2. for any feasible schedule $S_k(t)$ of μ_k, $W(\sigma_k(t)) \geq W(S_k(t))$.

In other words, an optimal schedule is a feasible schedule with minimum loss. There are possibly more than one optimal schedules for μ_k with finish time less than or equal to t. We donote by $\Sigma_k(t)$ the set of all of the optimal schedules for μ_k at t. Hence, if $S_k(t) \in \Sigma_k(t)$, $S_k(t)$ is an optimal schedule for μ_k at t.

The scheduling problem considered here is NP-complete. To prove that, its related *decision problem*, which is defined to be computing a feasible schedule with loss no more than a given bound, can be easily shown to be NP-complete. This can be done by restricting to PARTITION problem [1] by setting $r_i = 0, w_i = c_i, d_i = \frac{1}{2}\sum_{j=1}^{n} c_j$, for $1 \leq i \leq n$.

3 Scheduling a Permutation

We consider the problem of finding an optimal schedule for the task set in two steps – select a permutation, and find an optimal schedule for the permutation. The methodology is presented in Figure 1.

```
Loop 1: Choose a permutation $\mu$ of $\Gamma$
          Loop 2: for $\mu_k$, $k = 1, 2, \ldots, n$
                    Loop 3: compute $\sigma_k(t)$
```

Figure 1: Methodology

Clearly, to find the optimal schedule for the task set, all possible permutations have to be considered. How to search the permutations will be addressed in section 4. In Loop 3, optimal schedules for μ_k are computed at some time instants. Next, we discuss how to compute $\sigma_k(t)$ for a given t in the following, and then discuss how to determine the time instants for μ_k.

3.1 Computing $\sigma_k(t)$

We use dynamic programming to compute $\sigma_k(t)$ based on $\sigma_{k-1}(t')$, with $t' \leq t$. The criticality of τ_k plays an important role in computing $\sigma_k(t)$.

If τ_k is a critical task, we have to schedule it, possibly at the cost of rejecting some of the non-critical tasks. Hence, $\sigma_k(t) = S_{k-1}(t') \oplus \tau_k$, for some schedule $S_{k-1}(t')$, where $\oplus$ means concatenation of the sequence and the task. The finish time of $S_{k-1}(t')$ must be no more than $t - c_k$ in order to accommodate τ_k, which leads to $t' \leq t - c_k$. The best candidate could be $\sigma_{k-1}(t - c_k)$. Hence,

$$\sigma_k(t) = \sigma_{k-1}(t - c_k) \oplus \tau_k, \tag{1}$$

which can be seen in Figure 2. Note that $\sigma_k(t)$ only exists for a *proper* range of t. That is, $\sigma_k(t)$ is infeasible when t is beyond the proper range, e.g., $t < r_k + c_k$, or if $\sigma_{k-1}(t - c_k)$ is infeasible. The range would be considered in details later.

If τ_k is non-critical, our concern is to obtain as large a weight for the schedule as possible, while the critical tasks accepted previously must be kept in the schedule. Computation of $\sigma_k(t)$ is based

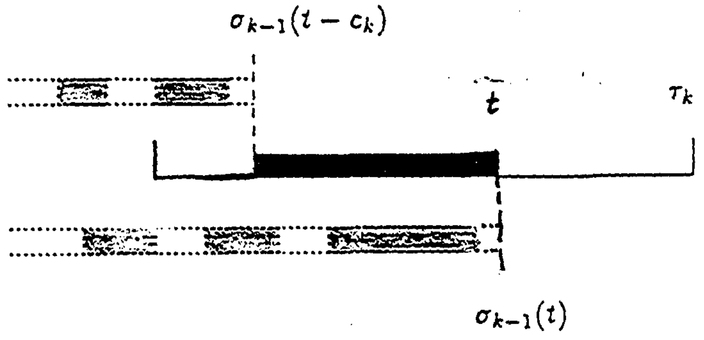


Figure 2: Scheduling for τ_k

upon the choice between either including τ_k or not. That is,

$$\sigma_k(t) = \begin{cases} \sigma_{k-1}(t-c_k) \oplus \tau_k & \text{or} \\ \sigma_{k-1}(t) \end{cases} \tag{2}$$

which can be seen in Figure 2. The factors for making the choice are the feasibility and the weights of the two candidate schedules. That is, the chosen schedule has to be feasible in the first place, and has a weight more than or equal to the other.

3.2 Time Instants for Computing $\sigma_k(t)$

From Equations 1 and 2, the computation of $\sigma_k(t)$ is based on the results of $\sigma_{k-1}(t)$ and $\sigma_{k-1}(t-c_k)$. We do not need to look for all possible values for t. We can get the idea about how to determine the time instants t by a simple example in Figure 3. The ready times, computation times, deadlines, and weights are given to the tasks in $\mu_3 = \langle\tau_1, \tau_2, \tau_3\rangle$.

The following schedules for μ_3 can be easily verified.

$\sigma_3(t) = \text{INFEASIBLE}$		for $t < 6$
$\sigma_3(t) = \langle\tau_3\rangle$	$W(\sigma_3(t)) = 0$	for $6 \leq t < 7.5$
$\sigma_3(t) = \langle\tau_2, \tau_3\rangle$	$W(\sigma_3(t)) = 5$	for $7.5 \leq t < 9$
$\sigma_3(t) = \langle\tau_1, \tau_3\rangle$	$W(\sigma_3(t)) = 10$	for $9 \leq t$

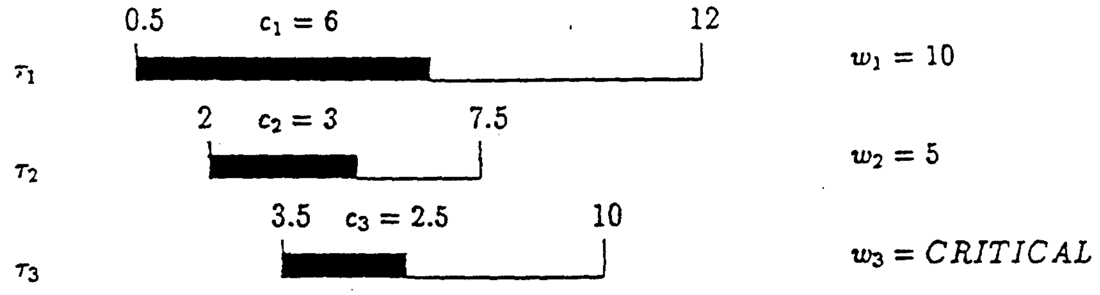


Figure 3: $\mu_3 = \langle \tau_1, \tau_2, \tau_3 \rangle$

In general, there exist a number of subranges in each of which the schedules are exactly identical, which are illustrated in Figure 4. We only need to compute the schedules at the time instants which delimit the subranges, i.e., 6, 7.5, and 9. We call these time instants *scheduling points*. The scheduling points can be determined by the timing characteristics of the tasks.

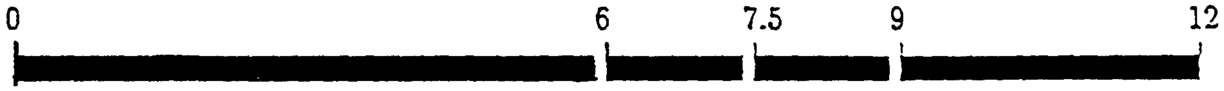


Figure 4: Identical subranges

3.3 Definition of Scheduling Points

We denote the jth scheduling point for μ_k by $\lambda_{k,j}$, and call j the *index* of $\lambda_{k,j}$. Hence, $\sigma_k(\lambda_{k,j})$ denotes an optimal schedule for μ_k at the scheduling point $\lambda_{k,j}$. Let v_k be the total number of scheduling points at which we need to schedule μ_k. For simplicity, λ_k denotes the set of $\lambda_{k,1}, \lambda_{k,2}, \ldots, \lambda_{k,v_k}$, and σ_k the set of $\sigma_k(\lambda_{k,1}), \sigma_k(\lambda_{k,2}), \ldots, \sigma_k(\lambda_{k,v_k})$. The scheduling points are defined as follows.

Definition: The set of scheduling points, λ_k, is *complete* if and only if:

1. for any $t < \lambda_{k,1}$, $\Sigma_k(t)$ is empty,
2. for any $\lambda_{k,j} \leq t < \lambda_{k,j+1}$, for $j = 1, \ldots, v_k - 1$, $\sigma_k(\lambda_{k,j}) \in \Sigma_k(t)$, and
3. for any $t \geq \lambda_{k,v_k}$, $\sigma_k(\lambda_{k,v_k}) \in \Sigma_k(t)$.

Note that $\Sigma_k(t)$ being empty means that there is no feasible schedule with finish time less than or equal to t. And also remember that $\sigma_k(\lambda_{k,j}) \in \Sigma_k(t)$ means that $\sigma_k(\lambda_{k,j})$ is an optimal schedule for μ_k at t. The completeness of scheduling points indicates that all of the optimal schedules at the positive real time domain can be represented by the optimal schedules computed at the scheduling points. In addition, the set of scheduling points, λ_k, is *minimum*, if and only if $W(\sigma_k(\lambda_{k,j})) < W(\sigma_k(\lambda_{k,j+1}))$, for any $1 \leq j \leq v_k - 1$. This ensures that there does not exist any redundant scheduling point which, if removed, does not violate the completeness of the scheduling points. The sets of scheduling points that we will discuss are complete and minimum.

3.4 An Example for Deriving Scheduling Points

The values of λ_k depend on the temporal relations between r_k and λ_{k-1}. The example in Figure 5 is used to illustrate the relations. We only describe the idea of deriving scheduling points by the example, and will discuss in more details later. Assume that there are 5 scheduling points for μ_{k-1}, and we consider to compute σ_k based on σ_{k-1}. The current task, τ_k, may be critical or non-critical.

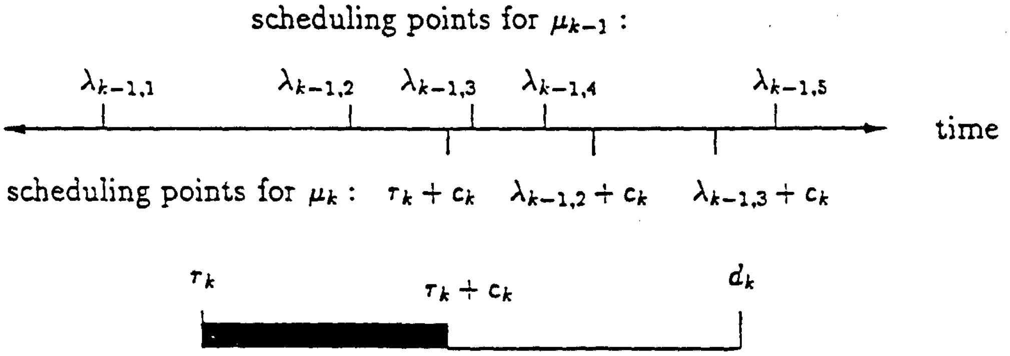


Figure 5: Scheduling Points

First, let us assume that τ_k is critical, which means that τ_k must be the last task in any feasible schedules for μ_k. A schedule for μ_k is thus a schedule for μ_{k-1} concatenated by τ_k. Hence, the optimal schedules for μ_k can be computed by appending τ_k to $\sigma_{k-1}(j)$, $j = 1, \ldots, v_{k-1}$. One restriction is that τ_k must be able to execute during its time window, from r_k to d_k. Hence, the scheduling points are $\lambda_{k-1,j} + c_k$, $j = 1, \ldots, v_{k-1}$, subject to the timing constraint of τ_k. In the example, because $r_k > \lambda_{k-1,1}$, the first scheduling point is $\lambda_{k,1} = r_k + c_k$. The first and the rest scheduling points are expressed in Equations 3-5. Notice that $\lambda_{k-1,4} + c_k > d_k$. Hence, there are only 3 scheduling points for μ_k.

$$\lambda_{k,1} = r_k + c_k \quad \text{and} \quad \sigma_k(\lambda_{k,1}) = \sigma_{k-1}(\lambda_{k-1,1}) \oplus \tau_k \tag{3}$$

$$\lambda_{k,2} = \lambda_{k-1,2} + c_k \quad \text{and} \quad \sigma_k(\lambda_{k,2}) = \sigma_{k-1}(\lambda_{k-1,2}) \oplus \tau_k \tag{4}$$

$$\lambda_{k,3} = \lambda_{k-1,3} + c_k \quad \text{and} \quad \sigma_k(\lambda_{k,3}) = \sigma_{k-1}(\lambda_{k-1,3}) \oplus \tau_k \tag{5}$$

On the other hand, let us assume that τ_k is non-critical. As a non-critical task, τ_k is not necessarily included in the schedule of μ_k. Whether to include τ_k or not depends on how much weight may be gained by including τ_k. If τ_k is included in the schedules, the new possible scheduling points for μ_k are expressed in Equations 6–8.

$$\lambda'_{k,1} = r_k + c_k \quad \text{and} \quad \sigma'_k(\lambda'_{k,1}) = \sigma_{k-1}(\lambda_{k-1,1}) \oplus \tau_k \tag{6}$$

$$\lambda'_{k,2} = \lambda_{k-1,2} + c_k \quad \text{and} \quad \sigma'_k(\lambda'_{k,2}) = \sigma_{k-1}(\lambda_{k-1,2}) \oplus \tau_k \tag{7}$$

$$\lambda'_{k,3} = \lambda_{k-1,3} + c_k \quad \text{and} \quad \sigma'_k(\lambda'_{k,3}) = \sigma_{k-1}(\lambda_{k-1,3}) \oplus \tau_k \tag{8}$$

If τ_k is not included, the scheduling points for μ_k are $\lambda_{k-1,j}$, $j = 1, \ldots, v_{k-1}$. The scheduling points for μ_k can be derived by, first, merging and sorting λ'_k and λ_{k-1}, which gives

$$\lambda_{k-1,1}, \lambda_{k-1,2}, \lambda'_{k,1}, \lambda_{k-1,3}, \lambda_{k-1,4}, \lambda'_{k,2}, \lambda'_{k,3}, \lambda_{k-1,5}. \tag{9}$$

Then, the resultant array of scheduling points should follow the rule that the weights of the optimal schedules at the scheduling points in the resultant array in Equation 9 should be strictly increasing. We remove any scheduling point if necessary.

3.5 Deriving Scheduling Points

By the example illustrated in Figure 5, λ_k can be derived from λ_{k-1} and τ_k. Note that a scheduling point indicates the finish time of a schedule. If we want to append τ_k to $\sigma_{k-1}(\lambda_{k-1,j})$, τ_k can not be started before $\lambda_{k-1,j}$. This implies that λ_k can be determined by the temporal relations between λ_{k-1}, the finish times of σ_k, and the start time of τ_k. Specifically, we need to explore the temporal relations between the earliest start time, r_k, the latest start time, $d_k - c_k$, of τ_k, and the lower and upper bounds to be defined below. We define the lower bound $L_{k-1} \equiv \lambda_{k-1,1}$, and the upper bound $U_{k-1} \equiv \lambda_{k-1,v_{k-1}}$. In particular, they have the following meanings.

L_{k-1}: the largest time instant such that there is no feasible schedule for μ_{k-1} with finish time less than L_{k-1}.

U_{k-1}: the least time instant such that the optimal schedule for μ_{k-1} with finish time greater than U_{k-1} can be $\sigma_{k-1}(\lambda_{k-1,v_{k-1}})$.

The six possible temporal relations in Equations 10–15 can be used to determine λ_k.

$$d_k - c_k < L_{k-1} \leq U_{k-1} \tag{10}$$

$$\tau_k \leq L_{k-1} \leq d_k - c_k < U_{k-1} \tag{11}$$

$$L_{k-1} < \tau_k \leq d_k - c_k < U_{k-1} \tag{12}$$

$$\tau_k \leq L_{k-1} \leq U_{k-1} \leq d_k - c_k \tag{13}$$

$$L_{k-1} < \tau_k \leq U_{k-1} \leq d_k - c_k \tag{14}$$

$$L_{k-1} \leq U_{k-1} < \tau_k \tag{15}$$

The temporal relations are illustrated in Figure 6, and can be summarized in three cases. The method for constructing scheduling points according to the temporal relations is discussed next. The correctness of the method, i.e., the completeness and minimization of the scheduling points , is verified later.

3.5.1 τ_k is Critical

The task τ_k must be the last task in any feasible schedule of μ_k. Remember that $\sigma_k(t)$ can be computed by Equation 1. In the following, we discuss how to derive the scheduling points for the three cases. The readers may refer to the algorithm in section 3.7 for details.

Case 1 $d_k - c_k < L_{k-1}$: μ is not feasible. Remember that there exists no feasible schedule for μ_k with finish time less than L_{k-1}, due to the completeness of scheduling points, and that $d_k - c_k$ is the latest start time for τ_k. Hence, μ_k is not feasible, and thus the whole permutation, μ, is not feasible.

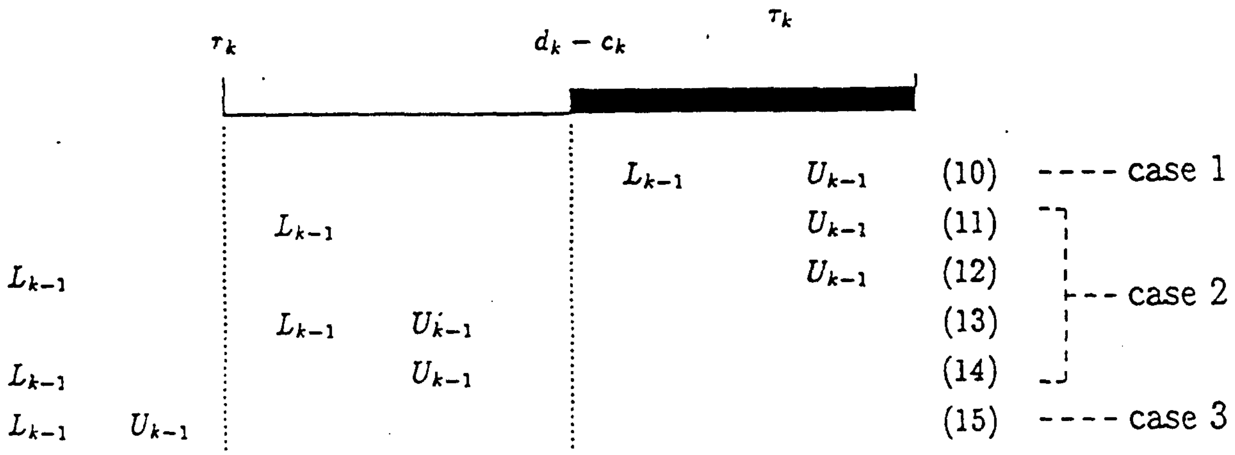

r_k
$d_k - c_k$
r_k
L_{k-1} U_{k-1} (10) ---- case 1
L_{k-1} U_{k-1} (11)
L_{k-1} U_{k-1} (12) ---- case 2
L_{k-1} U_{k-1} (13)
L_{k-1} U_{k-1} (14)
L_{k-1} U_{k-1} (15) ---- case 3

Figure 6: Temporal relations

Case 2 ($r_k \leq L_{k-1} \leq d_k - c_k$) or ($L_{k-1} < r_k \leq U_{k-1}$) : The scheduling points for μ_k is the set of $\lambda_{k-1,j} + c_k$, $j = 1, \ldots, v_{k-1}$, subject to the constraints that τ_k must start after r_k, and finish before d_k. Specifically, λ_k can be derived by Equations 16 and 17.

$$\lambda_{k,1} = max(\lambda_{k-1,1} + c_k, r_k + c_k) \tag{16}$$

Let J_{min} and J_{max} denote the smallest and the largest integers of j satisfying $\lambda_{k,1} < \lambda_{k-1,j} + c_k \leq d_k$. The rest of the scheduling points can be computed by

$$\lambda_{k,i} = \lambda_{k-1,j} + c_k, \quad where\ J_{min} \leq j \leq J_{max}\ and\ i = j - J_{min} + 2 \tag{17}$$

Note that $v_k = J_{max} - J_{min} + 2$. The example given in Figure 5 falls in this case.

Case 3 $U_{k-1} < r_k$: there is only one scheduling point. Since r_k is the earliest start time for τ_k, the only scheduling point is $r_k + c_k$.

3.5.2 τ_k is Non-critical

Remember that $\sigma_k(t)$ can be computed by Equation 2. The non-critical task τ_k is not necessarily included in the schedule for μ_k. Whether to include τ_k or not depends on how much weight may be gained by including τ_k. Let us consider the three cases.

Case 1 $d_k - c_k < L_{k-1}$: do nothing. The latest start time of τ_k is less than the lower bound, L_{k-1}; hence, τ_k can not be included in any feasible schedule. The scheduling points and schedules for μ_{k-1} remain the same as the scheduling points and schedules for μ_k. In our implementation, to save time and space, λ_{k-1} and λ_k use the same memory spaces; also, σ_{k-1} and σ_k use the same memory spaces. So now $\lambda_k = \lambda_{k-1}$ and $\sigma_k = \sigma_{k-1}$.

Case 2 $(r_k \leq L_{k-1} \leq d_k - c_k)$ or $(L_{k-1} < r_k \leq U_{k-1})$: If τ_k is included, the new *possible* scheduling points for μ_k is the set of $\lambda_{k-1,j} + c_k$, $j = 1, \ldots, v_{k-1}$, subject to the constraints that τ_k must start after r_k, and finish before d_k. Specifically, the new possible scheduling points , λ'_k, can be derived by Equations 18 and 19.

$$\lambda'_{k,1} = max(\lambda_{k-1,1} + c_k, r_k + c_k) \tag{18}$$

Let J_{min} and J_{max} denote the smallest and the largest integers of j satisfying $\lambda'_{k,1} < \lambda_{k-1,j} + c_k \leq d_k$. The rest of the scheduling points are

$$\lambda'_{k,i} = \lambda_{k-1,j} + c_k, \quad where\ J_{min} \leq j \leq J_{max}\ and\ i = j - J_{min} + 2 \tag{19}$$

If τ_k is not included, the scheduling points for μ_k are the old ones for μ_{k-1}; i.e.,

$$\lambda_{k-1,j},\ j = 1, \ldots, v_{k-1}. \tag{20}$$

It is worth mentioning that some optimal schedules may include τ_k, and some may not. The scheduling points, λ_k, can be derived by the following two steps.

1. Merge and sort the two arrays of scheduling points, λ'_k and λ_{k-1}, in Equations 18-20.

2. The resultant array of scheduling points should follow the rule that the weights of the optimal schedules at the scheduling points should be strictly increasing. We remove any scheduling point that has a smaller weight than that of its preceding scheduling point in the array.

The example given in Figure 5 falls in this case.

Case 3 $U_{k-1} < r_k$: add one more scheduling point. The earliest start time of τ_k is greater than the upper bound, U_{k-1}; hence, the new scheduling point is $r_k + c_k$. The weight of the optimal schedule computed at this scheduling point is $W(\sigma_{k-1}(\lambda_{k-1,v_{k-1}})) + w_k$, which is larger than

$W(\sigma_{k-1}(\lambda_{k-1,v_{k-1}}))$. So this scheduling point must be included to make the set of scheduling points for μ_k complete. Note again that the scheduling points and schedules for μ_{k-1} remain unchanged as the scheduling points and schedules for μ_k; i.e., $\lambda_{k,j} = \lambda_{k-1,j}$ and $\sigma_k(\lambda_{k,j}) = \sigma_{k-1}(\lambda_{k-1,j})$, for $j = 1, \ldots, v_{k-1}$. However, $\lambda_{k,v_k} = r_k + c_k$ and $\sigma_k(\lambda_{k,v_k}) = \sigma_{k-1}(\lambda_{k-1,v_{k-1}}) \oplus \tau_k$, where $v_k = v_{k-1} + 1$.

3.6 Completeness and Minimization of Scheduling Points

We would like to show that the sets of scheduling points derived in the three cases are complete and minimum. Note that cases 1 and 3 are special cases, and are not difficult to verify. Hence, we will only briefly discuss case 2. If τ_k is critical, we would like to show that If λ_{k-1} is complete and minimum, λ_k derived by Equations 16 and 17 is also complete and minimum.

Condition 1 of completeness: Due to the completeness of λ_{k-1}, $\Sigma_{k-1}(t)$ is empty when $t < \lambda_{k-1,1}$. Equivalently, $\Sigma_{k-1}(t - c_k)$ is empty when $t < \lambda_{k-1,1} + c_k$. According to Equation 1, $\sigma_k(t) = \sigma_{k-1}(t - c_k) \oplus \tau_k$. Hence, $\sigma_k(t)$ does not exist when $t < \lambda_{k-1,1} + c_k$. On the other hand, since τ_k is critical, $\sigma_k(t)$ does not exist when $t < r_k + c_k$, which is the earliest finish time of τ_k. Therefore, $\Sigma_k(t)$ is empty when $t < \lambda_{k,1}$. This shows that condition 1 of the definition of completeness is satisfied.

Condition 2 of completeness: Due to the completeness of λ_{k-1}, $\sigma_{k-1}(\lambda_{k-1,j}) \in \Sigma_{k-1}(t)$, for any $\lambda_{k-1,j} \le t < \lambda_{k-1,j+1}$. By Equation 1, $\sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k$ is an optimal schedule at $\lambda_{k-1,j} + c_k$ for μ_k. Hence, $\sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k \in \Sigma_k(t)$, for $\lambda_{k-1,j} + c_k \le t < \lambda_{k-1,j+1} + c_k$. By Equation 17, $\lambda_{k,i} = \lambda_{k-1,j} + c_k$, for $i = j - J_{min} + 2$, which indicates that $\sigma_k(\lambda_{k,i}) = \sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k$. Besides, $\lambda_{k,i+1} = \lambda_{k-1,j+1} + c_k$, for $i + 1 = j + 1 - J_{min} + 2$, by Equation 17. Therefore, $\sigma_k(\lambda_{k,i}) \in \Sigma_k(t)$, for $\lambda_{k,i} \le t < \lambda_{k,i+1}$. This shows that condition 2 of the definition of completeness is satisfied.

Condition 3 of completeness: We know that $v_k = J_{max} - J_{min} + 2$. By Equation 17, $\lambda_{k,v_k} = \lambda_{k-1,J_{max}} + c_k$, which indicates that $\sigma_k(\lambda_{k,v_k}) = \sigma_{k-1}(\lambda_{k-1,J_{max}}) \oplus \tau_k$. Due to the completeness of λ_{k-1}, $\sigma_{k-1}(\lambda_{k-1,J_{max}}) \in \Sigma_{k-1}(t)$, for $\lambda_{k-1,J_{max}} \le t < \lambda_{k-1,J_{max}+1}$, or just $\lambda_{k-1,J_{max}} \le t$ if $J_{max} = v_{k-1}$. By Equation 1, $\sigma_{k-1}(\lambda_{k-1,J_{max}}) \oplus \tau_k$ is an optimal schedule at $\lambda_{k-1,J_{max}} + c_k$ for μ_k. Hence, $\sigma_{k-1}(\lambda_{k-1,J_{max}}) \oplus \tau_k \in \Sigma_k(t)$, for $\lambda_{k-1,J_{max}} + c_k \le t$. Note that the range of

$t < \lambda_{k-1,J_{max}+1} + c_k$ is removed. Because J_{max} is the largest integer of j satisfying $\lambda_{k-1,j} + c_k \le d_k$, the schedule $\sigma_{k-1}(\lambda_{k-1,J_{max}+1}) \oplus \tau_k$ would not be feasible. Since $\sigma_k(\lambda_{k,v_k}) = \sigma_{k-1}(\lambda_{k-1,J_{max}}) \oplus \tau_k$, $\sigma_k(\lambda_{k,v_k}) \in \Sigma_k(t)$ for $\lambda_{k,v_k} \le t$. This shows that condition 3 of the definition of completeness is satisfied.

Minimization: By Equation 1, $W(\sigma_k(t)) = W(\sigma_{k-1}(t - c_k) \oplus \tau_k) = W(\sigma_{k-1}(t - c_k))$, since a critical task has no weight. Because λ_{k-1} is minimum, $W(\sigma_{k-1}(\lambda_{k-1,j})) < W(\sigma_{k-1}(\lambda_{k-1,j+1}))$, for any $1 \le j \le v_{k-1} - 1$. That is, $W(\sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k) < W(\sigma_{k-1}(\lambda_{k-1,j+1}) \oplus \tau_k)$, for any $1 \le j \le v_{k-1} - 1$. By Equations 16 and 17, $W(\sigma_k(\lambda_{k-1,j} + c_k)) < W(\sigma_k(\lambda_{k-1,j+1} + c_k))$, and thus $W(\sigma_k(\lambda_{k,i})) < W(\sigma_k(\lambda_{k,i+1}))$, for any $1 \le i \le v_k - 1$. This shows that λ_k is minimum.

If τ_k is non-critical, τ_k may be included or not included in the optimal schedules for μ_k. Assuming that τ_k is not included in any of the optimal schedules, $\lambda_k = \lambda_{k-1}$ is complete, since λ_{k-1} is complete. However, including τ_k may gain some more weight, so we also need to consider the schedules including τ_k. If τ_k is included in the optimal schedules, λ'_k derived by Equations 18 and 19 is the complete set of scheduling points for the optimal schedules including τ_k, by the same reason described for the critical task. Hence, it is sufficient to construct the complete set of λ_k by selecting from λ'_k and λ_{k-1}. Since whether to include τ_k or not does not affect the feasibility of the schedules, we only need to consider the weights of the optimal schedules. A complete set of scheduling points indicates that the weights of the optimal schedules at these scheduling points should be non-decreasing. Furthermore, a complete and minimum set of scheduling points indicates that the weights of the optimal schedules at these scheduling points should be strictly increasing. Hence, we can merge and sort the two arrays of λ'_k and λ_{k-1}, and remove any scheduling point that has a smaller weight than that of its preceding scheduling point in the array. The resultant scheduling points is thus complete and minimum.

3.7 The Permutation Scheduling Algorithm (PSA)

Algorithm PSA:

```
Input: a permutation sequence $\mu = \langle \tau_1, \tau_2, \ldots, \tau_n \rangle$
Output: an optimal schedule $\sigma_n(\lambda_{n,v_n})$

Initialization: $v_0 = 1$; $\lambda_{0,1} = 0$; $\sigma_0(\lambda_{0,1}) = ()$; $W(\sigma_0(\lambda_{0,1})) = 0$
for $k = 1$ to $n$
    when $\boxed{\tau_k \text{ is critical}}$
        case 1 $(d_k - c_k < L_{k-1})$ : ($\mu$ is not feasible)
            exit

        case 2 $(\tau_k \leq L_{k-1} \leq d_k - c_k)$ or $(L_{k-1} < \tau_k \leq U_{k-1})$ :
            Computation for the first scheduling point:
                $\lambda_{k,1} = max(\lambda_{k-1,1} + c_k, \tau_k + c_k)$
                $j = 1$ if $\lambda_{k-1,1} > \tau_k$; otherwise, $j$ is the greatest integer such that $\lambda_{k-1,j} \leq \tau_k$
                $\sigma_k(\lambda_{k,1}) = \sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k$
                $W(\sigma_k(\lambda_{k,1})) = W(\sigma_{k-1}(\lambda_{k-1,j}))$
            Loop: $j = J_{min}$ to $J_{max}$, where $J_{min}$ and $J_{max}$ denote the smallest and the largest
                                        integers of $j$ satisfying $\lambda_{k,1} < \lambda_{k-1,j} + c_k \leq d_k$.
                $i = j - J_{min} + 2$
                $\lambda_{k,i} = \lambda_{k-1,j} + c_k$
                $\sigma_k(\lambda_{k,i}) = \sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k$
                $W(\sigma_k(\lambda_{k,i})) = W(\sigma_{k-1}(\lambda_{k-1,j}))$
            $v_k = J_{max} - J_{min} + 2$

        case 3 $(U_{k-1} < \tau_k)$ : (only one scheduling point )
            $\lambda_{k,1} = \tau_k + c_k$
            $\sigma_k(\lambda_{k,1}) = \sigma_{k-1}(\lambda_{k-1,v_{k-1}}) \oplus \tau_k$
            $W(\sigma_k(\lambda_{k,1})) = W(\sigma_{k-1}(\lambda_{k-1,v_{k-1}}))$
            $v_k = 1$
```

```
when $\boxed{\tau_k \text{ is non-critical}}$
    case 1 $(d_k - c_k < L_{k-1})$ : (scheduling points and schedules remain the same)
        /* Do nothing; $\tau_k$ cannot be included in any feasible schedule */
        /* Hence, $\lambda_k = \lambda_{k-1}$ and $\sigma_k = \sigma_{k-1}$ */

    case 2 $(\tau_k \leq L_{k-1} \leq d_k - c_k)$ or $(L_{k-1} < \tau_k \leq U_{k-1})$ :
        Computation for the first new possible scheduling point:
            $\lambda'_{k,1} = max(\lambda_{k-1,1} + c_k, \tau_k + c_k)$
            $j = 1$ if $\lambda_{k-1,1} > \tau_k$; otherwise, $j$ is the greatest integer such that $\lambda_{k-1,j} \leq \tau_k$
            $\sigma'_k(\lambda'_{k,1}) = \sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k$
            $W(\sigma'_k(\lambda'_{k,1})) = W(\sigma_{k-1}(\lambda_{k-1,j})) + w_k$
        Loop: $j = J_{min}$ to $J_{max}$, where $J_{min}$ and $J_{max}$ denote the smallest and the largest
                                    integers of $j$ satisfying $\lambda'_{k,1} < \lambda_{k-1,j} + c_k \leq d_k$.
            $i = j - J_{min} + 2$
            $\lambda'_{k,i} = \lambda_{k-1,j} + c_k$
            $\sigma'_k(\lambda'_{k,i}) = \sigma_{k-1}(\lambda_{k-1,j}) \oplus \tau_k$
            $W(\sigma'_k(\lambda'_{k,i})) = W(\sigma_{k-1}(\lambda_{k-1,j})) + w_k$
        construct $\sigma_k$ from $\sigma_{k-1}$ and $\sigma'_k$ by
            1) merging and sorting $\lambda_{k-1}$ and $\lambda'_k$ into one array
            2) making the weights of the schedules in the resultant array strictly
               increasing; removing any schedule off the array if necessary.

    case 3 $(U_{k-1} < \tau_k)$ : (adding one more scheduling point)
        $v_k = v_{k-1} + 1$
        $\lambda_{k,v_k} = \tau_k + c_k$
        $\sigma_k(\lambda_{k,v_k}) = \sigma_{k-1}(\lambda_{k-1,v_{k-1}}) \oplus \tau_k$
        $W(\sigma_k(\lambda_{k,v_k})) = W(\sigma_{k-1}(\lambda_{k-1,v_{k-1}})) + w_k$
        /* Note that $\lambda_{k,j} = \lambda_{k-1,j}$ and $\sigma_k(\lambda_{k,j}) = \sigma_{k-1}(\lambda_{k-1,j})$   for $j = 1$ to $v_{k-1}$ */
endfor
```

4 Scheduling a Task Set

To find an optimal schedule for the task set, we may have to consider all possible ($n!$) permutations. It is possible to reduce the search space by eliminating some infeasible permutations. For example, if $d_i < r_j$, there is no feasible schedule in which τ_i is placed after τ_j. Even after the reduction, the search space might still be too large. We propose to use *simulated annealing* technique, recognizing that while this technique reduces the search, it may yield sub-optimal results.

4.1 Simulated Annealing

Simulated annealing is a stochastic approach for solving large optimization problems. It was developed using statistical mechanics ideas to find a global minimum point in the energy space. Kirkpatrick *et al* [5] had demonstrated the power and applications of simulated annealing to the field of combinatorial optimization.

To find the optimal solution of the optimization problem is similar to finding the lowest energy state of metal. The metal is melted first. Then it is cooled down slowly until the freezing point is reached. At each temperature, a number of trials are carried out to reach the equilibrium. The temperature has to be controlled not to drop too quick; otherwise, it is possible to be trapped in a local minimum energy configuration. Lower energy generally indicates a better solution. The annealing process starts from a randomly chosen configuration, proceeding to seek potentially promising neighbor configurations. The neighbor configuration is derived by perturbing the current configuration. If the neighbor configuration has a lower energy, the change is always accepted. The distinct feature is that the neighbor configuration with a higher energy can also be accepted with the probability of $e^{(E-E')/T}$, where T is the temperature, and $E - E'$ represents the difference in the energy of current and neighbor configurations. Notice that when the temperature is high, an energy *up jump* is more likely than it is when the temperature is low, as it may reach the configuration, although with higher energy, which may lead to a better solution. An *up jump* means a jump from low energy to high energy, and a *down jump* means a jump from high energy to low energy.

4.2 The Set Scheduling Algorithm (SSA)

A permutation is used to represent the configuration. If a permutation is ordered in an Earliest Deadline First (EDF) fashion, we call it an EDF permutation. An EDF permutation may be a good starting permutation for the process of simulated annealing for this problem. If the window of a task is contained in the window of another task, we say that the latter task contains the former task. If there are no containing relations among tasks, the EDF permutation is a permutation of which an optimal schedule of the task set is a subsequence [4]. Thus, an optimal schedule for the task set can be generated by PSA by scheduling the EDF permutation. The energy function can be expressed by a loss function:

$$loss = \sum weight\ of\ rejected\ noncritical\ tasks$$

A schedule is not acceptable if critical tasks are rejected. We may say that the loss of a rejected critical task is infinity. However, this kind of assignment makes it difficult to distinguish between a very bad schedule (e.g., a critical task is rejected) and even a worse schedule (more critical tasks are rejected). In general, the former schedule can be considered as an improvement over the latter one. If the loss incurred by a rejected critical task is assigned infinity, there is no way to tell which is better between the schedule in which one critical task is rejected and that in which three critical tasks are rejected. Hence, we assign a finite amount of loss to rejected critical tasks. The loss of a critical task must be large enough such that the scheduler will not reject a critical task to accommodate a number of non-critical tasks.

The neighbor function may be obtained using one of the following two methods. In the first, simple method, we randomly select one task from those rejected. This task is inserted in a randomly chosen location within a specified distance from its original location, where the *distance* is the number of tasks between two tasks in a permutation. The distance is used in this approach to control the degree of perturbation.

The reason of rejecting a task is due to the acceptance of other tasks. Given a schedule for a permutation, it is sometimes difficult to identify which task results in the rejection of other tasks, especially when tasks are congested together. However, the task immediately before or after those rejected is likely to play a role. In the second method, we try to identify the task which causes the largest loss of weight. As a simple approach, we attribute the rejection of a task to the task accepted prior to it. Then we choose the task which causes the largest loss of weight and insert it within a specified distance. Due to the robustness of simulated annealing technique, the impact of not necessarily selecting the task which caused the largest loss is minimal. Note that in simulated annealing many parameters are randomized, and the energy function, together with the temperature, control the progress of the annealing process. Tindell *et al* [9] commented that the great beauty of the simulated annealing lies in that you only need to describe what constitutes a good solution without worrying about how to reach it. According to our experiments, we find that the first method performs better than the second method. However, the process in the first method sometimes falls into a local minimum. The combination of the two methods does perform better than any of the individual one. The Set Scheduling Algorithm (SSA) is presented in Figure 7.

The initial temperature has to be large enough such that virtually all up jumps are allowed in the beginning of the annealing process. According to [9], the way to compute new temperature is that new temperature = α * current temperature, where $0 \leq \alpha \leq 1$. A step denotes an iteration in the inner loop in Figure 7, which is the process of scheduling a permutation and determining whether the permutation would become the current permutation. The thermal equilibrium can be reached if a certain number of down jumps or a certain number of total steps has been observed; and the freezing point, or the stopping condition, can be reached if no further down jump has been observed in a certain number of steps [5, 9].

```
Algorithm SSA:

Begin
    choose initial temperature $T$
    choose edf permutation as the starting permutaion, $\mu$
    schedule $\mu$ by PSA and compute its energy, $E$
    loop
        loop
            compute neighbor permutation $\mu'$
            schedule $\mu'$ by PSA and compute its energy, $E'$
            if $E' < E$ then
              making $\mu'$ the current permutation: $\mu \leftarrow \mu'$ and $E \leftarrow E'$
            else
              if $e^{\frac{E-E'}{T}} \geq$ random(0,1) then
                making $\mu'$ the current permutation: $\mu \leftarrow \mu'$ and $E \leftarrow E'$
              else
                $\mu$ remains as the current permutation
        until thermal equilibrium is reached
        compute new temperature: $T \leftarrow \alpha * T$
    until stopping condition is reached
End
```

Figure 7: Set Scheduling Algorithm

5 Experiment Result

Experiments **are conducted to study** the performance of SSA based on:

- scheduling ability $= \frac{\text{number of times that the algorithm generates a feasible schedule}}{\text{number of times that there does exist a feasible schedule for the task set}}$
- loss ratio $= \frac{\text{loss of the schedule generated by SSA} - \text{loss of an optimal schedule}}{\text{total weight of accepted noncritical tasks of an optimal schedule}}$
- iterations = number of permutations that the simulated annealing algorithm goes through to obtain the sub-optimal schedule

We start with an EDF permutation. To study how good the result would be by using PSA to schedule the EDF permutation, the scheduling ability and loss ratio for the EDF permutation are computed as well. In our experiments, a task set consists of 100 tasks. The number of permutations in such a task set is $100! \approx 9.33 * 10^{157}$. To study how good the output of SSA is compared to an optimal schedule, it is rather impractical to go through such a great number of permutations for a task set to derive the optimal schedule and its minimum loss for comparison. Instead, we choose to make up a task set such that the task set is feasible and the loss of its optimal schedule is 0. Although the SSA algorithm is primarily designed for an overloaded system, we apply SSA to such task sets for measuring the performance. The parameters are shown in Figure 8.

parameters	*value*	*type*
window length	mean_Wl = 20.0	truncated normal distribution
computation time	mean_C = $\frac{mean\_Wl}{3}$	truncated normal distribution
load	20%, 40%, 60%, 80%	constants
criticality ratio	25%, 50%, 75%	constants
weight	low_W=1, high W=50	discrete uniform distribution

Figure 8: Parameters of the experiments

The mean of window length, mean_Wl, is set to be 20 time units. The load is the ratio of total computation time to the largest deadline, D, in the task set. Hence, the load indicates the difficulty of scheduling the task set. The mean of computation time, mean_C, is one third of the mean of window length, which allows the windows among tasks to overlap to some extent. How much the windows overlap partially depends on the load. If the load is high, the windows are congested together, and thus the overlapping is high. We expect some containing relations between tasks to occur and thus increase the difficulty for scheduling. Note that, without containing relations, scheduling the task set would be straightforward. The standard deviations of window length and computation time are set to be their means, respectively. Criticality ratio indicates the percentage of the critical tasks in the task set. It is set to be 25%, 50%, and 75%. The higher the criticality ratio, the more difficult it is to generate a feasible schedule for the task set. On the other hand, although it is easier to come up with a feasible schedule when the criticality ratio is low, the loss ratio may still be high. It may be necessary to go through many permutations before an acceptable loss ratio is reached. In our experiments, the acceptable loss ratio is set to be 0%, which means that SSA will keep trying different permutations until either the loss ratio is 0 or the stopping condition is reached, in which SSA fails to find an optimal schedule. Note that a big energy (loss), 1000, is incurred for a rejected critical task. Hence, for an infeasible schedule, the loss ratio may well be more than 100%. The weight of a non-critical task is an integer ranging from low_W=1 to high_W=50, determined by a discrete uniform distribution function. For each individual experiment with different parameters, 200 task sets, each with 100 tasks, are generated for scheduling. The way of creating a feasible task set without loss is described in appendix A.

From Figure 9a, The scheduling ability of SSA is 98.5% when criticality ratio is 75% and load is 80%, and is 100% for other lower criticality ratios and loads. This is because the simulated annealing algorithm focuses on searching suitable neighbor permutations in such a way that the rejected critical tasks, if any, may be accepted. Note that scheduling only the EDF permutation can not always generate a feasible schedule. The scheduling ability of scheduling EDF permutation degrades when load increases, which means tasks congest more together. The scheduling ability of scheduling EDF permutation also degrades when the criticality ratio increases, which makes meeting the deadlines of all critical tasks become more difficult.

As far as non-critical tasks are concerned, SSA can not guarantee the minimum loss. However, even in the worst case given in Figure 9b, the loss ratio is less than 10%. The loss ratio becomes less when criticality ratio or load is less. In many cases, the loss ratios are less than 5%. As for scheduling the EDF permutation, the loss ratios are significantly larger.

The number of permutations to be searched in simulated annealing depends on the situations of energy jumps, the way of reducing temperature, and how we define thermal equilibrium and stopping conditions. In the experiments, we find that reducing temperature faster does not impose a negative impact on the scheduling ability and loss. How to set the parameters in simulated annealing differs a great deal from one application to another. We do want to generate the result as good as possible, but are not willing to spend more computation time than necessary. This usually requires fine tuning the parameters to get the trade-off between the two goals. We find that the following parameters are beneficial: initial temperature = 3000, $\alpha = 0.8$ (instead of 0.95 or even 0.99 suggested in other applications), the number of down jumps to obtain thermal equilibrium = 25, the number of total steps to obtain thermal equilibrium = 300, the number of steps with no further down jump to obtain the freezing point = 2000, which is also the stopping condition. The average number of permutations searched in simulates annealing is given in Figure 9c. If SSA can successfully generate a feasible schedule, the average number of permutations checked is no more than 4000 times. The number increases a little if SSA fails to find a feasible schedule, because in this case SSA does not stop until the freezing point is reached. Note that the average numbers of permutations are less than n^2, which can roughly give us the idea about the complexity of searching over the permutation space. Additional studies have shown that if we modify the above parameters to increase the average number of permutations by about 10 times, the loss ratios can be further reduced by about 25% of the loss ratios obtained here.

If time can be expressed in integers, the dynamic programming technique used in PSA can be applied by computing $\sigma_k(t)$ at $t = 1, \ldots, D$. Let us call this approach the integral PSA, compared to the original PSA with scheduling points, denoted by PSA SP in Figures 9d. Obviously, the integral PSA tends to compute more schedules than the original PSA. We would like to see how more efficient the original PSA algorithm is than the integral PSA. Specifically, we compare the average

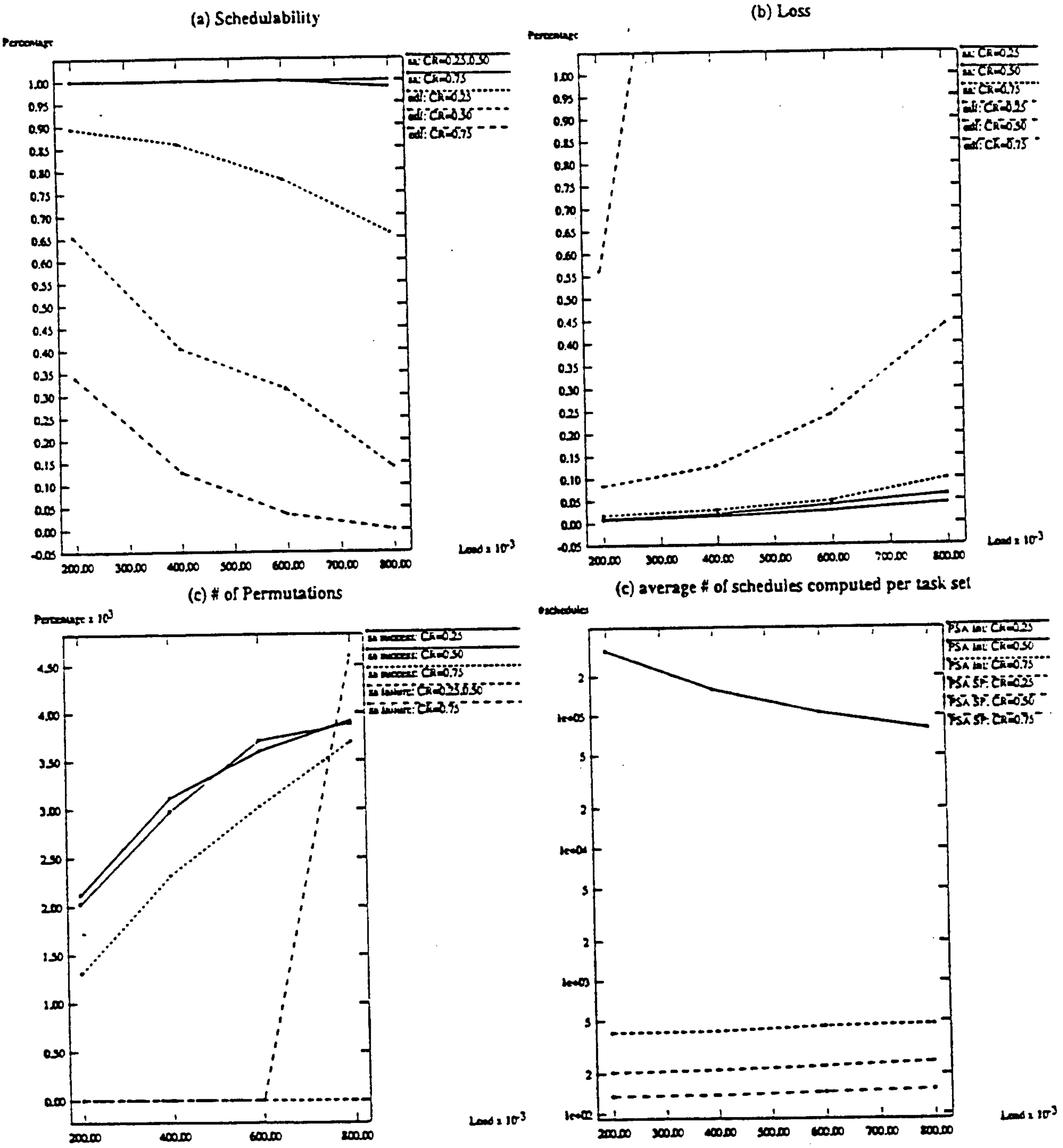


Figure 9: (a) Schedulability; (b) Loss; (c) #Permutations; (d) #Schedules

number of schedules required to derive the optimal schedule for a permutation. For the integral PSA, the number of schedules computed is fixed, or $n*D$, as can be seen in Figure 1. For the original PSA, $\sum_{k=1}^{n} v_k$ is the number of schedules needed to schedule a permutation. The average number of schedules needed to schedule a permutation by PSA is computed over the permutations of a task set, and is presented in Figure 9d. The number for the original PSA decreases with the criticality ratio. This is because a critical task never increases the number of scheduling points; instead, the number of scheduling points might be decreased due to the timing constraint of the critical task. For the criticality ratios of 0.25, 0.50, and 0.75, the average number of schedules required for a task set of 100 tasks are approximately 480, 250, and 150, respectively. The complexity of the original PSA seems linear in this sense. On the other hand, the complexity of the integral PSA is quite high. The number decreases with load. This happens to be related to the way of generating the task set, in which D = total_c / load. The number is equal to $n * D$, where D might fluctuate a little.

6 Conclusion

In this paper, we study the scheduling problem for a real-time system which is overloaded. A significant performance degradation may be observed in the system if the overload problem is not addressed properly [2]. As not all the tasks can be processed, the set of tasks selected for processing is crucial for the proper operation of an overloaded system. We assign to the tasks *criticalities and weights* on the basis of which the tasks are selected. The objective is to generate an optimal schedule for the task set such that all of the critical tasks are accepted, and then the loss of weights of non-critical tasks is minimum.

We present a two step process for generating a schedule. First, we develop a schedule for a permutation of tasks using a pseudo-polynomial algorithm. The concept of *scheduling points* is proposed for the algorithm. In order to find the optimal schedule for the task set, we have to consider all permutations. The simulated annealing technique is used to limit the search space while obtaining optimal or near optimal results. Our experimental results indicate that the approach is very efficient.

The work presented in this paper can be easily extended to address the overload issue for periodic tasks. To schedule a set of periodic tasks with criticalities and weights, we can convert the periodic tasks in the time frame of the least common multiple of the task periods to aperiodic tasks. The schedule generated for the frame can be applied repeatedly for the subsequent time frames.

Our algorithm can also be applied to solving the problem of scheduling imprecise computations [7], in which a task is decomposed logically into a mandatory subtask, which must finish before the deadline, and an optional subtask, which may not finish. The goal is to find a schedule such that the mandatory subtasks can all be finished by their deadlines and the sum of the computation times of the unfinished optional subtasks is minimum. A schedule satisfies the *0/1 constraint* if every optional subtask is either completed or discarded [7]. We can solve this problem by using our algorithm by setting the mandatory subtasks to be critical, and the optional subtasks to be non-critical with weights equal to their computation times.

Appendix A. Generating a task set

```
Generate computation times for tasks according to mean_C and the standard deviation
D = (total computation time) / load
Assigning starting instants, $s_k$, to tasks such that
        the intervals between the computation times are truncated normally distributed
For each task $\tau_k$
    Determine the criticality by criticality_ratio and/or weight by low_W and high_W
    Compute the window length of $\tau_k$ according to mean_Wl and the standard deviation
            (note that window length $\geq c_k$)
    align the window with the computation time in their middle points:
        $r_k = max(0, s_k + \frac{c_k}{2} - \frac{\mathrm{window\_length}}{2})$
        $d_k = min(D, r_k + \mathrm{window\_length})$
```

The load determines how the tasks would be congested. Once the largest deadline, D, has been computed, we separate the computation times of the tasks in such a way that the *positions* of the computation times on the time axis stretches over the range from 0 to D. Note that the starting instants of the computation times consist in an optimal schedule for the task set. In this way, all of the tasks in the task set can be accepted. At last, the windows are aligned with the computation times.

References

[1] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman Company, San Francisco, 1979.

[2] Jayant R. Haritsa, Miron Livny, and Michael J. Carey. Earliest deadline scheduling for real-time database systems. In *IEEE Real-Time Systems Symposium*, Dec. 1991.

[3] Shyh-In Hwang, Sheng-Tzong Cheng, and Ashok K. Agrawala. An optimal solution for scheduling real-time tasks with rejection. In *International Computer Symposium*, Dec. 1994.

[4] Shyh-In Hwang, Sheng-Tzong Cheng, and Ashok K. Agrawala. Optimization in non-preemptive scheduling for aperiodic tasks. Technical Report CS-TR-3216, UMIACS-TR-94-14, Department of Computer Science, University of Maryland at College Park, Jan. 1994.

[5] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science(220)*, pages 671–680, 1983.

[6] C. L. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[7] W.K. Shih, J. Liu, and J.Y. Chung. Fast algorithms for scheduling imprecise computations. In *IEEE Real-Time Systems Symposium*, pages 12–19, Dec. 1989.

[8] Philip Thambidurai and Kishor S. Trivedi. Transient overloads in fault-tolerant real-time systems. In *IEEE Real-Time Systems Symposium*, Dec. 1989.

[9] K.W. Tindell, A. Burns, and A.J. Wellings. Allocating hard real-time tasks: An np-hard problem made easy. *The Journal of Real-Time Systems*, 4(2):145–165, June 1992.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE November 1994	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE Scheduling an Overloaded Real-Time System	5. FUNDING NUMBERS N00014-91-C-0195 DASG-60-92-C-0055
6. AUTHOR(S) Shyh-In Hwang, Chia-Mei Chen and Ashok K. Agrawala	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Maryland Department of Computer Science A. V. Willliams Building College Park, MD 20742	8. PERFORMING ORGANIZATION REPORT NUMBER CS-TR-3377 UMIACS-TR-94-128
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Honeywell, Inc. 3600 Technology Drive Minneapolis, MN 55418 Phillips Laboratory Directorate of Contracting 3651 Lowry Avenue SE Kirtland AFB NM 87117-5777	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The real-time systems differ from the conventional systems in that every task in the real-time system has a timng constraint. Failure to execute the tasks under the timing constraints may result in fatal errors. Sometimes, it may be impossible to execute all the tasks in the task set under their timing constraints. Considering a system with limited resources, one solution to handle the overload problem is to reject some of the tasks in order to generate a feasible schedule for the rest. In this paper, we consider the problem of scheduling a set of tasks without preemption in which each task is assigned criticality and weight. The goal is to generate an optimal schedule such that all of the critical tasks are scheduled and then the non-critical tasks are included so that the weight of rejected non-critical tasks is minimized. We consider the problem of finding the optimal schedule in two steps. First, we select a permutation sequence of the task set. Secondly, a pseudo-polynomial algorithm is proposed to generate an optimal schedule for the permutation sequence. If the global optimal is desired, all permutation sequences have to be considered. Instead, we propose to incorporate the simulated annealing technique to deal with the large search space. Our experimental results show that our algorithm is able to generate near optimal schedules for the task sets in most cases while considering only a limited number of permutations.

14. SUBJECT TERMS Process Management; Nonnumerical Algorithms and Problems	15. NUMBER OF PAGES 29
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

Notes on Symbol Dynamics[*][†]

Ashok K. Agrawala
Department of Computer Science, University of Maryland
College Park, Maryland 20742
E-mail: agrawala@cs.umd.edu

Christopher Landauer
System Planning and Development Division, The Aerospace Corporation
The Hallmark Building, Suite 187, 13873 Park Center Road, Herndon, Virginia 22071
Phone: (703) 318-1666, FAX: (703) 318-5409
E-mail: cal@aero.org

13 February 1995

Abstract

This paper introduces a new formulation of dynamic systems that subsumes both the classical discrete and differential equation models as well as current trends in hybrid models. The key idea is to express the system dynamics using symbols to which the notion of time is explicitly attached. The state of the system is described using symbols which are active for a defined period of time. The system dynamics is then represented as relations between the symbolic representations.

We describe the notation and give several examples of its use.

[*] This work is supported in part by ONR and DARPA under contract N00014-91-C-0195 to Honeywell and Computer Science Department at the University of Maryland. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, ONR, the U.S. Government or Honeywell.
Computer facilities were provided in part by NSF grant CCR-8811954.

[†] This work is supported in part by ARPA and Philips Labs under contract DASG60-92-0055 to Department of Computer Science, University of Maryland. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, PL, or the U.S. Government.

Contents

1 Introduction

Traditionally, systems have been modelled using state variables defined in a metric space and the system dynamics defined using differential equations. This approach uses continuous descriptions of space and time. When we use computers for expressing and manipulating such models we have to use symbols to represent it. Symbols are discrete by their very nature, and require use of mapping from the continuous spaces to discrete spaces. These mappings cause problems unless carried out rather carefully. Further, when we consider the problems in which some aspects of the system are genuinely discrete, hybrid models have been used. As different techniques have to be used for continuous and discrete aspects of the system, significant complexity gets added to such models.

Recognizing that the computer systems only use symbols for any representations, in this paper we present a formulation of system dynamics directly in terms of symbols. In order to handle the synamics, time interval over which a symbol is considered valid is explicitly attached. The symbols describing different aspects of the system may be from a set appropriate for that aspect. The dynamics is described in terms of rules connecting the symbolic representations.

This paper contains the preliminary formulation of system dynamics in the framework of *Symbol Dynamics.*

2 Descriptions of System Behavior

For the purposes of this paper, *behavior* includes all the relationships among parts of a system at the same or different times. In particular, the combined relationships among parts of a system at the same time is usually called *structure*. Both of these aspects are subsumed in our use of the term behavior.

We assume that our ability to generate or derive new information about the system behavior changes only at discrete points in time, since we expect to perform these processes on digital computers. The event times define the time scale. In this paper, we introduce *Symbol Dynamics*, a totally symbolic way to represent the important aspects of dynamical systems and processes, so that we can reason about them using computers.

3 Concepts and Notations

This section contains the basic notions of Symbol Dynamics.

3.1 State Variable

We assume that systems exist and change over time. We are looking for a method of describing those changes so we can compute how to control them.

The systems we consider can be described with state variables. Each state variable is an observation on the system or a derivation from other state variables.

We may or may not know a priori which state variables are important, or even which ones are determinable (i.e., the system comes first, and the state variables are chosen to be helpful in describing the behavior). We might call the state variables *attributes* of the state.

3.2 Symbol

We want to measure and compute with information about a system, so we need to map the system into formal spaces we understand better.

A *type* is a symbol set, both representing a set of values and including some operations on those values; this is the notion of formal space used here. It includes collections of mutually dependent types and functions between different types.

A *symbol* of a given type is an element of the set of values that type. Any notions of credibility, confidence, or uncertainty are part of the type system that is used. It is especially important to define the allowable operations on these kinds of types. For example, for measurements of a system, the symbol would include the measured value and the associated uncertainty value.

3.3 Attribute Identifier

We assume that we will want to know different things about the system behavior. We need names to keep track of the different things we measure or compute.

An *attribute identifier* is a name for a state variable (a state variable is like a probe into some aspect of the system behavior, and the attribute identifier is only the label).

3.4 Expression

An *expression* is a pair
(attribute identifier: symbol),
which is interpreted to mean the assertion that the state variable can be described by the symbol (when the expression is active). We will describe the precise semantics of these expressions later on.
These are models of the state variable values.

3.5 Interval

An *interval* is a pair
[start time, end time),
assumed to describe a half-open interval (to save us from trouble with the topology). The end time may be omitted, in which case it is interpreted to mean infinity by default.

3.6 Characterizer

A *characterizer* is a pair
(expression, interval),
also written
(attribute identifier: symbol; start time, end time),
interpreted to mean that the expression is *active* during the specified interval. It becomes active at the start time, and becomes inactive at the end time. Each characterizer has a *range* (its *interval of activity*) and a *scope* (the set of attribute identifiers that occur in its expression).
We may also consider a symbol set that includes arithmetic expressions that contain an explicit time variable t. For example,
$(p : p_0 + v_0 * t; t_0, t_1)$
represents a continuous change along the interval.
We will also have occasion to reason about conditions at particular points in time, so the assertion language will also have characterizers of the form
(expression, point).

3.7 Event

An *event* is the activation or deactivation of a characterizer. We make no limiting assumptions about simultaneous events.

4 System Description

A *system description* is a finite set of characterizers, so we assume explicitly that a system can be described by a finite set of characterizers. We insist that only a finite set of characterizers be active at any one time. Since each of those characterizers is active over a positive interval, there is therefore some small interval thereafter during which all of them are still active.
Everything we know about a system's behavior is described by characterizers and relationships among the characterizers. Domain models and context can be written as characterizers, generally with large intervals.

4.1 Dynamics

Relationships among characterizers are rules that define the dynamics. These rules take the form:

> if *these* characterizers (with a list) are active on *these* intervals, then *this* new one is also active on *this* other interval (not necessarily contained in the intersection of the original intervals).

Rules can contain variable identifiers, with implicit universal quantification.

Relationships hold on intervals and the combination may extend the range. We generate new characterizers according to the relationships, either predictive (range extension) or deductive (knowledge extension).

The language in which the rules are written is important, since it has to accommodate notations from many different types, many of which will not be known when the language is defined. Some basic concepts that will be in any of these languages are continuity and derivatives.

It is important to remember that the system comes first, and that the state variables are our choices for modeling and understanding the system. This means in particular that the coordinate systems we use are temporary, and that the constraints among the state variables are expressed explicitly as relationships.

4.2 Normalization and Continuation

Characterizers may have overlapping intervals. *Normalization* is the process of breaking each characterizer into two or more others, to fit the time scale. If t is an event time, and

$$(a : v; s, e)$$

is a characterizer with $s < t \leq e$, then we can replace it with two characterizers

$(a : v; s, t)$ and $(a : v; t, e)$.

If two characterizers use the same attribute,

$$(a : v; s, e)$$

and

$$(a : w; t, u),$$

then we say that the second one *continues* the first one iff they are adjacent in time, so $t = e$. Continuity considerations in the transition from v to w at time t are treated in the next section.

In any system with a finite density of event times, if we split every characterizer that spans an event time, then we end up with characterizers that start and stop at consecutive event times (though they may be *continued* by other characterizers). This has some computational conveniences.

If we have two characterizers

$$(a : v; t_1, t_2)$$

and

$$(a : w; t_2, t_3),$$

so that the second one continues the first, then we need some kind of explicit characterizer for the transition, active in an interval containing the transition time. If there is a description u in an appropriate domain for which

$$u = \begin{cases} v, & \text{for } t_1 \leq t < t_2, \\ w, & \text{for } t_2 \leq t < t_3, \end{cases}$$

then we can conclude

$$(a : u; t_1, t_3).$$

This is the opposite of normalization.

If there is an overlap, that is, if the two characterizers

$$(a : v; t_1, t_2)$$

and

$$(a : w; t_3, t_4)$$

have

$[t_1, t_2) \cap [t_3, t_4)$ non-empty,

and

$v(t) = w(t)$ for $t \in [\max(t_1, t_3), \min(t_2, t_4))$,

then we can also conclude

$$(a : u; \min(t_1, t_2), \max(t_3, t_4)).$$

4.3 Continuation and Continuity

One aspect of continuity is transitions from one symbol to another across interval boundaries. The transition relations are extra conditions that have to hold *at* the transition time (usually they are smoothness conditions for model transitions).

A typical smoothness property is infinitesimal: for characterizers

$$(a : v; t_0, t_1)$$

and

$$(a : w; t_1, t_2),$$

we normally want smoothness, written

$$\left.\frac{\mathrm{d}\, v}{\mathrm{d}\, t}\right|_{t=t_1^-} = \left.\frac{\mathrm{d}\, w}{\mathrm{d}\, t}\right|_{t=t_1^+},$$

and continuity, written

$$v(t = t_1^-) = w(t = t_1^+).$$

Both of these are point conditions on the attributes and their derivatives, and we can consider only conditions on attributes by using whatever derivatives are needed in the conditions: instead of

$$(a : v; t_0, t_1),$$

we use

$$(a : (v, v'); t_0, t_1),$$

and write our smoothness condition as

$$\begin{pmatrix} v \\ v' \end{pmatrix}_{t=t_0^-} = \begin{pmatrix} w \\ w' \end{pmatrix}_{t=t_0^+}$$

If we also require continuity in each attribute, so that

$$w(t = t_1^+) = w(t = t_1),$$

then the upper limit in the previous expression can be omitted.

It is therefore clear that we must deal with point events at transitions

$$[t_0 ... t_1)\, [t_1 ... t_2),$$

but not with point characterizers. If we make the transition continuity a property of the definition of continuation, then we can assert it or not in any given model.

Of course, the expression $t = t_1^-$ means that the interval $[t_1 - \epsilon, t_1)$ is part of the limit computation for every ϵ small enough, so we might be able to use these intervals for some small enough ϵ without having to take the limits.

We will deal with these considerations in the simplest way possible. We have a characterizer that asserts continuity of the relevant attribute across a larger interval, such as $[t_0, t_2)$ above. The only place that the continuity characterizer has new information is at the transition point t_1, but we simply do not worry about the redundancy.

4.4 Characterizer Semantics and Inference

A characterizer is what we want to assume about what is true over its interval. It need not be consistent with the other characterizers in a system description; we explicitly allow false assertions here, so we can reason using counterfactuals.

4.4.1 Inference

We can make inferences within intervals, according to some rules. If, say, there is a rule

$$s_1 \& s_2 \Longrightarrow s_3,$$

and two characterizers

$$(v : s_1; t_0, t_1)$$

and

$$(v : s_2; t_2, t_3)$$

with $t_0 < t_2 < t_1 < t_3$, then we can conclude

$$(v : s_3; t_2, t_1).$$

4.4.2 Prediction

We can also make inferences that extend intervals in some cases. They take the form: If

$$(v : s_1; t_0, t_1)$$

and

$$(w : s_2; t_0, t_1)$$

are characterizers with $t_0 < t_1$, then there is a characterizer

$$(z : s_3; t_2, t_3)$$

for some t_2, t_3, with $t_0 < t_2 < t_1 < t_3$.

4.4.3 Truth Maintenance

Because we do not presume that the characterizers in a system are truths, we need to be much more careful about when they can be used together, especially in the inference and prediction processes. Since the inference rules themselves are time dependent, we need to keep track of the dependencies of every characterizer, both how and when it was derived (how tells us about hypotheses and inference rules; when helps us in checking temporal consistency) and its interval of activity.

We also need a way to indicate which characterizers we DO want to be true, so that different collections of characterizers can be compared and contrasted within the same context. We might want to consider computing various maximal consistent sets of irredundant assertions as an aid in this process.

Various rules can be activated that lead to new conclusions in an interval, which can supersede old ones; we also assume partial deduction, not total. We therefore need to use some kind of non-monotonic logic.

4.5 Analysis

Simulation is a continuing surprise.

We want tools with analytic power to help reduce our reliance on simulation, so we can make reliable predictions about the system behavior.

All of our computations are performed from the symbols active at a given time. The advantage of dealing explicitly with time in this formulation is that we can sit outside the usual sequencing of events, taking a kind of "side-long" look at the entire time line, and piece together parts of the models that we know more about regardless of whether or not they are the first ones in our time interval of interest.

We can also perform the deductions in an order that is different from the order imposed by time, using any of a number of simple mechanisms, such as rule-based systems or rewrite logics; both are being investigated.

5 Examples

This section contains several examples that illustrate the utility of the notation.

5.1 ODE

A simple example that shows range extension is an ordinary differential equation (ODE). For ODEs, the solution method is part of changing an ODE into a set of characterizers.

So let us consider a simple second-order ODE for the sine function,

$$\begin{aligned} y'' &= -y, \\ y'(0) &= 1, \\ y(0) &= 0, \end{aligned}$$

and solve it with Euler's method (a particularly bad one for this kind of problem, by the way).

First, we transform the equations into a first order system (in the usual way) by taking $x = y'$,

$$\begin{aligned} x' &= -y, \\ y' &= x, \\ x(0) &= 1, \\ y(0) &= 0, \end{aligned}$$

and we also define $z = x' = y''$.

5.1.1 First-Order

Now the way Euler's method works is by linear extrapolation, so for a given time $t = t_0$, if we have

$$\begin{aligned} x(t_0) &= x_0, \\ y(t_0) &= y_0, \end{aligned}$$

then we have

$$z_0 = z(t_0) = -y_0,$$

and we take

$$\begin{aligned} x(t) &= x_0 + z_0 * (t - t_0), \\ y(t) &= y_0 + x_0 * (t - t_0), \end{aligned}$$

for t in some small interval

$[t_0, t_1 = t_0 + dt)$.

The characterizers that describe this situation are:

$(x : x_0 + z_0 * (t - t_0); t_0, t_0 + dt)$,

$(y : y_0 + x_0 * (t - t_0); t_0, t_0 + dt)$,

which we want to be true for all choices of x_0, y_0, t_0, and dt (which ones we actually use in our system description depend on how we choose the time intervals in the solution).

The characterizers that describe the initial conditions are difficult, because they cannot be described with half-open intervals of the shape we have thus far described:

$(x : 1; 0)$,

$(y : 0; 0)$,

which is always going to be a problem in systems that start at a certain time.

In a more sophisticated system, the choice of next time interval would depend on the computed accuracy of the current solution.

For this example, we simply make all the time intervals the same, and say that the characterizer pair

$(x : x_1 + z_1 * (t - t_1); t_1, t_1 + dt)$,

$(y : y_1 + x_1 * (t - t_1); t_1, t_1 + dt)$

propagates the pair

$(x : x_0 + z_0 * (t - t_0); t_0, t_0 + dt)$,

$(y : y_0 + x_0 * (t - t_0); t_0, t_0 + dt)$

iff

$$\begin{aligned} x_1 &= x_0 + z_0 * dt, \\ y_1 &= y_0 + x_0 * dt, \\ t_1 &= t_0 + dt, \end{aligned}$$

which are the conditions for the first pair to meet the second (the condition $z_1 = -y_1$ is part of the definition of these characterizer pairs).

Extending the iteration, we have

$$\begin{aligned} x(0) &= 1, \\ y(0) &= 0, \\ x(k+1) &= x(k) - y(k) * dt, \\ y(k+1) &= y(k) + x(k) * dt, \end{aligned}$$

which can be written as a vector equation (we put the matrix on the right so we can use row vectors)

$$\begin{aligned} (x, y)(0) &= (1, 0), \\ (x, y)(k+1) &= (x, y)(k) \begin{pmatrix} 1 & dt \\ -dt & 1 \end{pmatrix}, \end{aligned}$$

so if we write I for the identity matrix and J for the matrix

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

then we have (with X = (x,y))

$$\begin{aligned} X(0) &= (1, 0), \\ X(k+1) &= X(k)(I + J * dt), \end{aligned}$$

so

$$X(k) = (1, 0)\ (I + J * dt)^{k},$$

which can be computed exactly.

Since the eigenvalues of $(I + J * dt)$ are $1 \pm i * dt$, which have magnitude $1 + dt^2$, the successive powers of the matrix diverge for any $dt > 0$, and therefore so does the iteration.

5.1.2 Second-Order Example

In this section, we use the same differential equation problem, with a different solver, a second-order one that is almost able to converge properly. We therefore have

$$x' = -y,$$

$$\begin{aligned} y' &= x, \\ x(0) &= 1, \\ y(0) &= 0, \end{aligned}$$

as above. Our initial conditions are

$(x : 1; 0)$,

$(y : 0; 0)$,

as before.

The method we use is a simplified second-order Runge-Kutta method [?], [?], which basically amounts to averaging the usual Euler approximation in an interval with a linear reapproximation at the endpoint of the interval. At a given time $t = t_0$, if we have

$$\begin{aligned} x(t_0) &= x_0, \\ y(t_0) &= y_0, \end{aligned}$$

then we have

$$\begin{aligned} x(t) &= x_0 - y_0 * dt - x_0 * dt^2/2, \\ y(t) &= y_0 + x_0 * dt - y_0 * dt^2/2, \end{aligned}$$

and it is the extra dt^2 terms that make the method second-order.

As above, we assume equal time intervals and get an iteration

$$\begin{aligned} x(0) &= 1, \\ y(0) &= 0, \\ x(k+1) &= x(k) - y(k) * dt - x(k) * dt^2/2, \\ y(k+1) &= y(k) + x(k) * dt - y(k) * dt^2/2, \end{aligned}$$

which can be written as a vector equation

$$\begin{aligned} (x,y)(0) &= (1,0), \\ (x,y)(k+1) &= (x,y)(k) \begin{pmatrix} 1 - dt^2/2 & dt \\ -dt & 1 - dt^2/2 \end{pmatrix}, \end{aligned}$$

and we have as above

$$\begin{aligned} X(0) &= (1,0), \\ X(k+1) &= X(k)(I * (1 - dt^2/2) + J * dt), \end{aligned}$$

so

$$X(k) = (1,0)\ (I * (1 - dt^2/2) + J * dt)^k,$$

which can be computed exactly.

Since the eigenvalues of $(I * (1 - dt^2/2) + J * dt)$ are $1 - dt^2/2 \pm i * dt$, which have magnitude $1 + dt^4/4$, this simple method still does not converge (but much more slowly).

5.1.3 Higher-Order Example

A similar analysis of the usual 4th-order Runge-Kutta method leads to an iteration

$$\begin{aligned} x(t) &= x_0 - y_0 * dt - x_0 * dt^2/2 + y_0 * dt^3/6 + x_0 * dt^4/24, \\ y(t) &= y_0 + x_0 * dt - y_0 * dt^2/2 - x_0 * dt^3/6 + y_0 * dt^4/24, \end{aligned}$$

with matrix

$$\begin{pmatrix} 1 - dt^2/2 + dt^4/24 & dt - dt^3/6 \\ -dt + dt^3/6 & 1 - dt^2/2 + dt^4/24 \end{pmatrix},$$

and eigenvalue magnitude of $1 + dt^6/36 + dt^8/24^2$, which is still greater than one. In fact, since this equation (in (x, y) space) represents moving around a circle, any extrapolation method based on tangents at a single point will fail, since all of the tangent vectors point outward from the circle. We note that the iteration equations do have the first terms of the usual Maclaurin series for $\sin(dt)$ and $\cos(dt)$, so we try out a different iteration:

$$\begin{aligned} x(t) &= x_0 * cos(dt) - y_0 * sin(dt), \\ y(t) &= y_0 * cos(dt) + x_0 * sin(dt), \end{aligned}$$

which can be written as a vector equation

$$(x,y)(0) = (1,0),$$

$$(x, y)(k+1) \;=\; (x, y)(k) \begin{pmatrix} cos(dt) & sin(dt) \\ -sin(dt) & cos(dt) \end{pmatrix},$$

and we have as above

$$\begin{aligned} X(0) &= (1, 0), \\ X(k+1) &= X(k)(I * cos(dt) + J * sin(dt)), \end{aligned}$$

so

$$\begin{aligned} X(k) &= (1, 0)\ (I * cos(dt) + J * sin(dt))^{k}, \\ &= (1, 0)\ (I * cos(k * dt) + J * sin(k * dt)), \end{aligned}$$

and

$$\begin{aligned} x(k * dt) &= cos(k * dt), \\ y(k * dt) &= sin(k * dt), \end{aligned}$$

from which we can hazard a guess as to the correct solution.

5.2 Measurement

Let us take a simple system in which the velocity and position are occasionally known through inexact measurement.
Our state variables are p for the position, v for the velocity, and a for the unknown acceleration.
We assume that the acceleration a is bounded by some constant A, so that for any times $t_0 < t_1$

$$|v(t_1) - v(t_0)| \;<\; |t_1 - t_0| * A.$$

We assume that we have characterizers

$$(a(t); t_{i-1}, t_i)$$

that describe the acceleration, and model characterizers

$$(v = p'; 0^-, -),$$
$$(c = v'; 0^-, -).$$

Therefore, we can compute the velocity and position by

$$v(t) = v(t_0) \;+\; \int_{t_0 < u < t} a(u)\ du,$$

$$p(t) = p(t_0) \;+\; \int_{t_0 < u < t} v(u)\ du.$$

The problem is to choose measurement times and variables that maintain a certain accuracy in the estimates of position.
We assume that we can measure position within a bound

$$|pmeas(t) - p(t)| \;<\; P,$$

and that we can measure velocity within a bound

$$|vmeas(t) - v(t)| \;<\; V,$$

but that we want to keep our estimate p_{est} of position either more accurately than the position measurement error (this might or might not be possible) or using as few measurements as possible.
We assume first that x_0, v_0 are known, and consider an interval $[t_0, t_1)$. We compute

$$|v(t_1) - v_0| \;<\; |t_1 - t_0| * A,$$

and therefore

$$|x(t_1) - x_0| \;<\; \frac{1}{2} * |t_1 - t_0|^2 * A,$$

so we would have to choose

$$\triangle\, t = t_1 - t_0$$

so that

$$\triangle\, t \;\le\; |V/A|$$

to keep the velocity within bounds, and

$$(\triangle\, t)^2 \;\le\; |2 * P/A|$$

to keep the position within bounds.
But of course, we don't know $x(t)$ or $v(t)$ after the first time interval, so we need to change the previous derivation a bit.
We assume that we know x_0 and v_0, and that

$$|x(t_0) - x_0| \;\le\; \triangle\, x_0$$

describes the accuracy of our knowledge of $x(t)$ at time $t = t_0$, and

$$|v(t_0) - v_0| \leq \Delta v_0$$

describes the accuracy of our knowledge of $v(t)$ at time $t = t_0$. Then the above inequalities become

$$|v(t_1) - v_0| \leq \Delta v_0 + |t_1 - t_0| * A,$$

and therefore

$$|x(t_1) - x_0| \leq \Delta x_0 + |t_1 - t_0| * \Delta v_0 + \frac{1}{2} * |t_1 - t_0|^2 * A,$$

so we have to have

$$\Delta t \leq |(V - \Delta v_0)/A|$$

to keep the velocity within bounds, and

$$(\Delta t)^2 + \frac{2 * \Delta v_0}{A} * (\Delta t) \leq |2 * (P - \Delta x_0)/A|$$

to keep the position within bounds.

At this point, we are stuck unless we can say something more helpful about the acceleration. Suppose we know that the acceleration jumps around, and that it has a distribution of values with mean 0 and variance R. In this case, we might be able to reduce the estimates for position and velocity and improve the time intervals.

References

[1] P. Henrici, Elements of Numerical Analysis, Wiley (1964)

[2] J. Stoer, R. Bulirsch, Einfu:hrung in die Numerische Mathematik, II Springer (1973)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
February 13, 1995

3. REPORT TYPE AND DATES COVERED
Technical Report

4. TITLE AND SUBTITLE
Notes on Symbol Dynamics

5. FUNDING NUMBERS
N00014-91-C-0195 and
DSAG-60-92-C-0055

6. AUTHOR(S)
Ashok K. Agrawala and Christopher Landauer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Maryland
A.V. Williams Building
College Park, Maryland 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
CS-TR - 3411
UMIACS-TR - 95-15

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Honeywell
3660 Technology Drive
Minneapolis, MN 55418

Phillips Labs
3550 Aberdeen Ave. SE
Kirtland AFB, NM
87117-5776

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This paper introduces a new formulation of dynamic systems that subsumes both the classical discrete and differential equation models as well as current trends in hybrid models. The key idea is to express the system dynamics using symbols to which the notion of time is explicitly attached. The state of the system is described using symbols which are active for a defined period of time. The system dynamics is then represented as relations between the symbolic representations. We describe the notation and give several examples of its use.

14. SUBJECT TERMS
C.m, Miscellaneous

15. NUMBER OF PAGES
11 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRA
Unlimited

Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints*

Sheng-Tzong Cheng and Ashok K. Agrawala
Institute for Advanced Computer Studies
Systems Design and Analysis Group
Department of Computer Science
University of Maryland
College Park, MD 20742
{stcheng,agrawala}@cs.umd.edu

Abstract

Allocation problem has always been one of the fundamental issues of building the applications in distributed computing systems (DCS). For real-time applications on DCS, the allocation problem should directly address the issues of task and communication scheduling. In this context, the allocation of tasks has to fully utilize the available processors and the scheduling of tasks has to meet the specified timing constraints. Clearly, the execution of tasks under the allocation and schedule has to satisfy the precedence, resources, and other synchronization constraints among them.

Recently, the timing requirements of the real-time systems emerge that the relative timing constraints are imposed on the consecutive executions of each task and the inter-task temporal relationships are specified across task periods. In this paper we consider the allocation and scheduling problem of the periodic tasks with such timing requirements. Given a set of periodic tasks, we consider the least common multiple (LCM) of the task periods. Each task is extended to several instances within the LCM. The scheduling window for each task instance is derived to satisfy the timing constraints. We develop a simulated annealing algorithm as the overall control algorithm. An example problem of the sanitized version of the Boeing 777 Aircraft Information Management System is solved by the algorithm. Experimental results show that the algorithm solves the problem in a reasonable time complexity.

*This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Honeywell or Army/Phillips.

1 Introduction

The task allocation and scheduling problem is one of the basic issues of building real-time applications on a distributed computing system (DCS). DCS is typically modeled as a collection of processors interconnected by a communication network. For hard real-time applications, the allocation of tasks over DCS is to fully utilize the available processors and the scheduling is to meet their timing constraints. Failure to meet the specified timing constraints or inability to respond correctly can result in disastrous consequence.

For the hard real-time applications, such as avionics systems and nuclear power systems, the approach to guarantee the critical timing constraints is to allocate and schedule tasks *a priori*. The essential solution is to find an static allocation in which there exists a feasible schedule for the given task sets. Ramamritham [Ram90] proposes a global view where the purpose of allocation should directly address the schedulability of processors and communication network. A heuristic approach is taken to determine an allocation and find a feasible schedule under the allocation. Tindell *et al.* [TBW92] take the same global view and exploit a simulated annealing technique to allocate periodic tasks. A distributed rate-monotonic scheduling algorithm is implemented. In each period a task must execute once before the specified deadline. The transmission times for the communications are taken into account by subtracting the total communication time from the deadline and making the execution of the task more stringent.

Simply assuring that one instance of each task starts after the ready time and completes before the specified deadline is not enough. Some real-time applications have more complicated timing constraints for the tasks. For example, the relative timing constraints may be imposed upon the consecutive executions of a task in which the scheduling of two consecutive executions of a periodic task must be separated by a minimum execution interval. Communication latency can be specified to make sure that the time difference between the completion of the sending task and the start of the receiving task does not exceed the specified value. The Boeing 777 Aircraft Information Management System is such an example [CDHC94]. For such applications, the algorithms proposed in literature do not work because the timing constraints are imposed across the periods of tasks. In this paper, we consider the relative timing constraints for real examples of real-time applications in Section 2. Based on the task characteristics, we propose the approach to allocate and schedule these applications in Section 3. A simulated annealing algorithm is developed to solve the problem in which the reduction on the search space is given in Section 4. In Section 5, we evaluate the practicality and show the significance of the algorithm. Instead of randomly generating the *ad hoc* test cases, we apply the algorithm to a real example. The example is the Boeing 777 AIMS with various numbers of processors. The experimental results are shown in Section 5.

2 Problem Description

Various kinds of periodic task models have been proposed to represent the real-time system characteristics. One of them is to model an application as an independent set of tasks, in which each task is executed once every period under the ready time and deadline constraints. Synchronization (e.g. precedence and mutual exclusion) and communications are simply ignored. Another model to take the precedence relationship and communications into account is to model the application as a task graph. In a task graph, tasks are represented as nodes while communications and precedence relationship between tasks are represented as edges. The absolute timing constraints can be imposed on the tasks. Tasks have to be allocated and scheduled to meet their ready time and deadline constraints upon the presence of synchronization and communications. The deficiency of task graph modeling is inability of specifying the relative constraints across task periods. For example, one can not specify the minimum separation interval between two consecutive executions of the same task.

In the work [CA93], we modified the real-time system characteristics by taking into account the relative constraints on the instances of a task. We considered the scheduling problem of the periodic tasks with the relative timing constraints. We analyzed the timing constraints and derive the scheduling window for each task instance. Based on the scheduling window, we presented the time-based approach of scheduling a task instance. The task instances are scheduled one by one based on their priorities assigned by the proposed algorithms. In this paper we augment the real-time system characteristics by considering the inter-task communication on DCS.

2.1 Task Characteristics

The problem considered in this chapter has the following characteristics.

- The Fundamentals: A task is denoted by the 4-tuple $< p_i, e_i, \lambda_i, \eta_i >$ denoting the period, computation time, low jitter and high jitter respectively. One instance of a task is executed each period. The execution of a task instance is non-preemptable. The start times of two consecutive instances of task τ_i are at least $p_i - \lambda_i$ and at most $p_i + \eta_i$ apart. Let s_i^j and f_i^j be the start time and finish time of task instance τ_i^j respectively. The timing constraints specified in Equations 1 through 4 must be satisfied.

$$f_i^j = s_i^j + e_i \tag{1}$$

$$s_i^{n_i+1} = s_i^1 + \text{LCM} \tag{2}$$

$$s_i^j \geq s_i^{j-1} + p_i - \lambda_i \tag{3}$$

$$s_i^j \leq s_i^{j-1} + p_i + \eta_i \tag{4}$$
$$\forall j = 2, \ldots, n_i + 1.$$

- **Asynchronous Communication:** Tasks communicate with each others by sending and receiving data or messages. The frequencies of sending and receiving tasks of a communication can be different. In consequence, communications between tasks may cross the task periods. When such asynchronous communications occur, the semantics of undersampling is assumed. When two tasks of different frequencies are communicating, schedule the message only at the lower rate. For example, if task A (of 10HZ) sends a message to task B (of 5HZ), then in every $200ms$, one of two instances of task A has to send a message to one instance of task B. If the sending and receiving tasks are assigned to the same processor, then a local communication occurs. We assume the time taken by a local communication is negligible. When an interprocessor communication (IPC) occurs, the communication must be scheduled on the communications network between the end of the sending task execution and the start of the receiving task execution. The transmission time required to communicate the message i over the network is denoted by μ_i.

- **Communication Latency:** Each communication is associated with a communication latency which specifies the maximum separation between the start time of the sending task and the completion time of the receiving task.

- **Cyclic Dependency:** Research on the allocation problem has usually focused on acyclic task graphs [Ram90, HS92]. Given an acyclic task graph $G = \{V, E\}$, if the edge from task A to task B is in E then the edge from B to A can not be in E. The use of acyclic task graphs excludes the possibility of specifying the cyclic dependency among tasks. For example, consider the following situation in which one instance of task A can not start its execution until it receives data from the last instance of task B. After the instance of task A finished its execution, it sends data to the next instance of task B. Since tasks A and B are periodic, the communication pattern goes on throughout the lifetime of the application. To be able to accommodate this situation, we take cyclic dependency into consideration.

The timing constraints described above are shown in Figure 1. For periodic tasks A and B, the start times of each and every instance of task execution and communication are pre-scheduled such that (1) the execution intervals fall into the range between $p - \lambda$ and $p + \eta$ and (2) the time window between the start time of sending task and the completion time of receiving task is less than the latency of the communication. In Figure 2, we illustrate examples of all possible communication patterns considered in this paper. The description of the communications in the task system is in the form of "*From sender-task-id (of frequency) To receiver-task-id (of frequency)*". If the sender

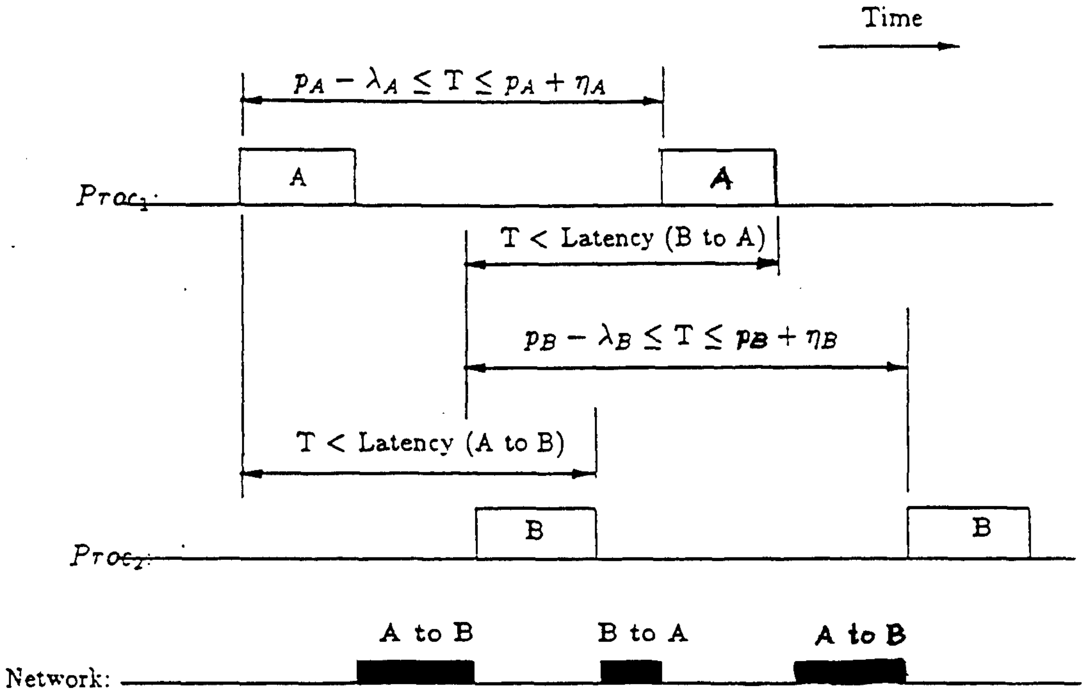
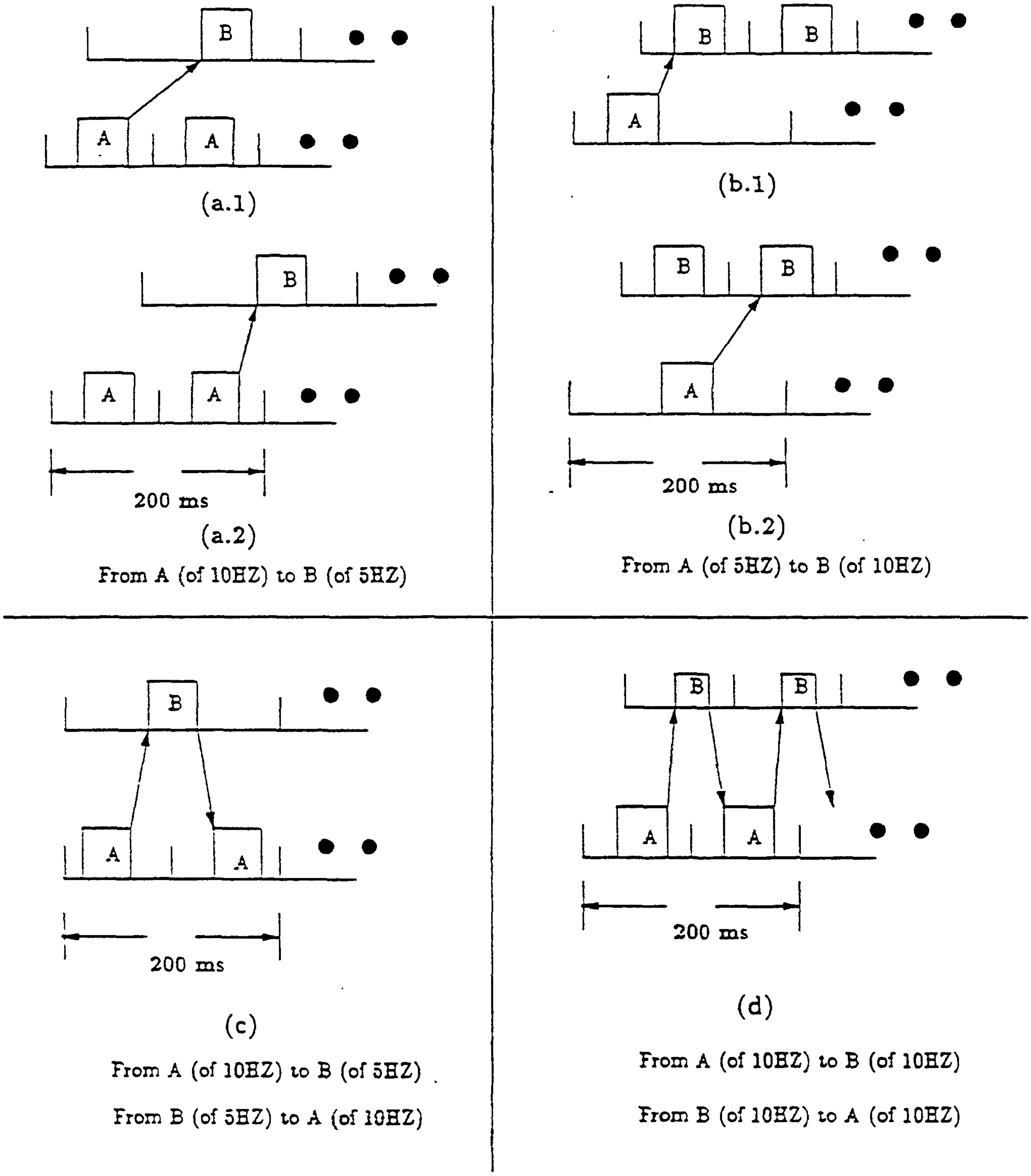


Figure 1: Relative Timing Constraints

frequency is n times of the receiver frequency and no cyclic dependency is involved, then one of every n instances of the sending task has to communicate with one instance of the receiving task. (Examples of this situation are shown in Figures 2.a.1 and 2.a.2. Likewise, for the case in which the receiver frequency is n time that of the sender frequency and no cyclic dependency is present, the patterns are shown in Figures 2.b.1 and 2.b.2. For an asynchronous communication, the sending (receiving) task in low frequency sends (receives) the message to (from) the *nearest* receiving (sending) task as shown in Figure 2.a (2.b). The cases where cyclic dependency is considered are shown in Figures 2.c and 2.d.

2.2 System Model

A real-time DCS consists of a number of processors connected together by a communications network. The execution of an instance on a processor is nonpreemptable. To provide predictable communication and to avoid contention for the communication channel at the run time, we make the following assumptions. (1) Each IPC occurs at the pre-scheduled time as the schedule is generated. (2) At most one communication can occur at any given time on the network.



(a.1)

(a.2)

From A (of 10HZ) to B (of 5HZ)

(b.1)

(b.2)

From A (of 5HZ) to B (of 10HZ)

(c)

From A (of 10HZ) to B (of 5HZ)

From B (of 5HZ) to A (of 10HZ)

(d)

From A (of 10HZ) to B (of 10HZ)

From B (of 10HZ) to A (of 10HZ)

Figure 2: Possible Communication Patterns

2.3 Problem Formulation

We consider the static assignment and scheduling in which a task is the finest granularity object of assignment and an instance is the unit of scheduling. We applied the simulated annealing algorithm [KGV83] to solve the problem of real-time periodic task assignment and scheduling with hybrid timing constraints. In order to make the execution of instances satisfy the specifications and meet the timing constraints, we consider a scheduling frame whose length is the least common multiple (LCM) of all periods of tasks. Given a task set Γ and its communications C, we construct a set of task instances, I, and a set of multiple communications, M. We extend each task $\tau_i \in \Gamma$ to n_i instances, $\tau_i^1, \tau_i^2, \ldots,$ and $\tau_i^{n_i}$. These n_i instances are added to I. Each communication $\tau_i \mapsto \tau_j \in C$ is extended to $\min(n_i, n_j)$[1] undersampled communications where $n_i = \text{LCM}/p_i$ and $n_j = \text{LCM}/p_j$. These multiple communications are added to M. The extension can be stated as follows.

- If $n_i < n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^1 \mapsto \tau_j^?, \tau_i^2 \mapsto \tau_j^?, \ldots,$ and $\tau_i^{n_i} \mapsto \tau_j^?$.
- If $n_i > n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^? \mapsto \tau_j^1, \tau_i^? \mapsto \tau_j^2, \ldots,$ and $\tau_i^? \mapsto \tau_j^{n_j}$.
- If $n_i = n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^1 \mapsto \tau_j^1, \tau_i^2 \mapsto \tau_j^2, \ldots,$ and $\tau_i^{n_i} \mapsto \tau_j^{n_j}$.

A task ID with a superscript of question mark indicates some instance of the task. For example, $\tau_i^1 \mapsto \tau_j^?$ means that τ_i^1 communicates with some instance of τ_j. We describe how we assign the nearest instance for each communication in Section 4.1.2.

The problem can be formulated as follows. Given a set of task instance, I, its communications M, we find an assignment ϕ, a total ordering σ_m of all instances, and a total ordering σ_c of all communications to minimize

$$\begin{aligned} E(\phi, \sigma_m, \sigma_c) &= \sum_{i,j} \delta(p_i - \lambda_i - s_i^{j+1} + s_i^j) + \sum_{i,j} \delta(s_i^{j+1} - s_i^j - p_i - \eta_i) \\ &+ \sum_{i,j} \delta(f_i^j - d_i^j) + \sum_{i,j,k,l} \delta(F(t_i^j \mapsto t_k^l, \sigma_c) - s_k^l) \\ &+ \sum_{i,j,k,l} \delta(f_k^l - s_i^j - \text{Latency } (\tau_i \text{ to } \tau_k)) \qquad (5) \\ &\text{subject to } \; s_i^j \geq r_i^j \text{ and } \; S(t_i^j \mapsto t_k^l, \sigma_c) \geq f_i^j, \;\; \forall\, t_i^j \mapsto t_k^l, \end{aligned}$$

where

[1] Due to undersampling, when an asynchronous communication is extended to multiple communications, the number of multiple communications is the smaller number of sender and receiver instances.

- s_i^j is the start time of τ_i^j under σ_m.

- f_i^j is the completion time of τ_i^j under σ_m.

- $r_i^j = p_i \times (j-1) + r_i$, and $d_i^j = p_i \times (j-1) + d_i$.

- $\delta(x) = 0$, if $x \leq 0$; and $= x$, if $x > 0$.

- $\phi(\tau_i)$ is the ID of processor which τ_i is assigned to.

- $\tau_i^j \mapsto t_k^l$ is the communication from τ_i^j to τ_k^l. If $\phi(\tau_i) = \phi(\tau_k)$, then $\tau_i^j \mapsto \tau_k^l$ is a local communication.

- $S(c, \sigma_c)$ is the start time of communication c on the network under σ_c.

- $F(c, \sigma_c)$ is the completion time of communication c on the network under σ_c.

The minimum value of $E(\phi, \sigma_m, \sigma_c)$ is zero. It occurs when the executions of all instances meet the jitter constraints and all communications meet their latency constraints. A feasible multiprocessor schedule can be obtained by collecting the values of s_i^j and f_i^j, $\forall$ i and j. Likewise, a feasible network schedule can be obtained from $S(c, \sigma_c)$s and $F(c, \sigma_c)$s.

Since the task system is asynchronous and the communication pattern could be in the form of cyclic dependency, we solve the problem of finding a feasible solution $(\phi, \sigma_m, \sigma_c)$ by exploiting the cyclic scheduling technique and embedding the technique into the simulated annealing algorithm.

3 The Approach

3.1 Bounds of a Scheduling Window

Define the scheduling window for a task instance as the time interval during which the task can start. Traditionally, the lower and upper bounds of the scheduling window for a task instance are called earliest start time (*est*) and latest start time (*lst*) respectively. These values are given and independent of the start times of the preceding instances.

We consider the scheduling of periodic tasks with relative timing constraints described in Equations 3 and 4. The scheduling window for a task instance is derived from the start times of its preceding instances. A *feasible* scheduling window for a task instance τ_i^j is a scheduling window in which any start time in the window makes the timing relation between s_i^{j-1} and s_i^j satisfy

Equations 3 and 4. Formally, given s_i^1, s_i^2, ..., and ..., s_i^{j-1}, the problem is to derive the feasible scheduling window for τ_i^j such that a feasible schedule can be obtained if τ_i^j is scheduled within the window.

Proposition 1 [CA93]: Let the est and lst of τ_i^j be

$$\begin{aligned} est(\tau_i^j) &= max\{(s_i^{j-1} + p_i - \lambda_i),\ (s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)\}, & (6) \\ \text{and } lst(\tau_i^j) &= min\{(s_i^{j-1} + p_i + \eta_i),\ (s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)\}. & (7) \end{aligned}$$

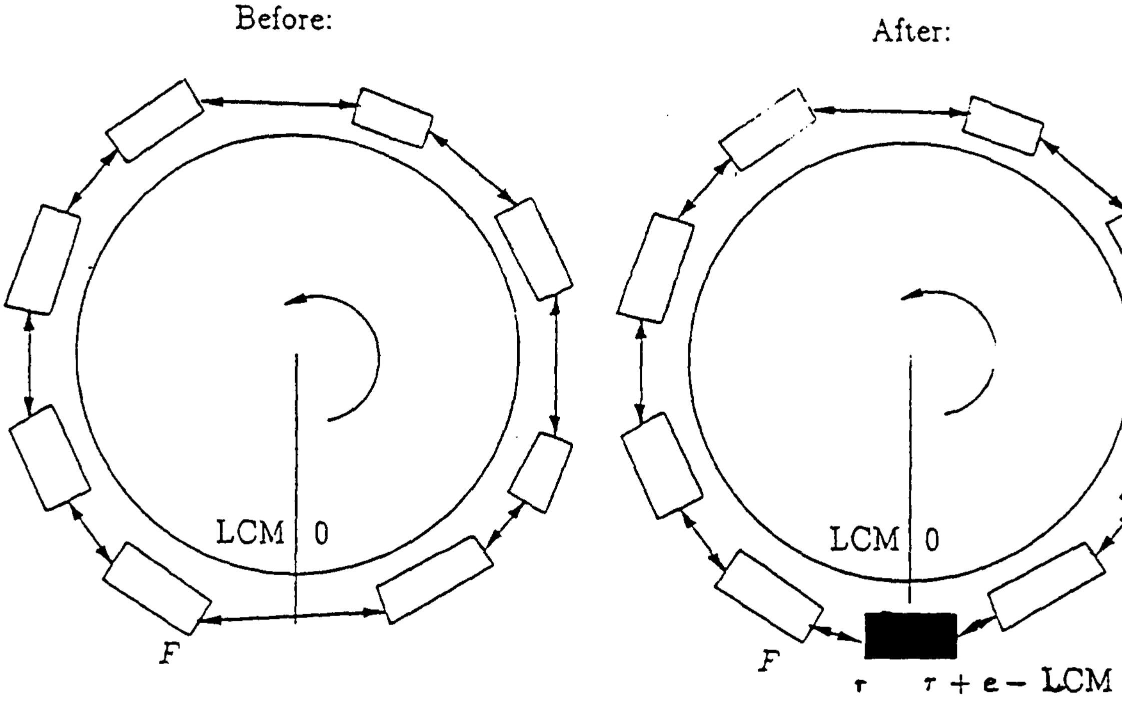
If s_i^j is in between the $est(\tau_i^j)$ and $lst(\tau_i^j)$, then the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible window.

3.2 Cyclic Scheduling Technique

The basic approach of scheduling a set of synchronous periodic tasks is to consider the execution of all instances within the scheduling frame whose length is the LCM of all periods. The release times of the first periods of all tasks are zero. As long as one instance is scheduled in each period within the frame and these executions meet the timing constraints, a feasible schedule is obtained. In a feasible schedule, all instances complete the executions before the LCM.

On the other hand, in asynchronous task systems, as depicted in Figure 2 in which the LCM is $200ms$, the periods of the two tasks are out of phase. It is possible that the completion time of some instance in a feasible schedule exceeds the LCM. To find a feasible schedule for such an asynchronous system, a technique of handling the time value which exceeds the LCM is proposed.

The technique is based on the linked list structure described in the work [CA93]. Without loss of generality, we assume the minimum release time among the first periods of all tasks is zero. We keep a linked list for each processor and a separated list for the communication network. Each element in the list represents a time slot assigned to some instance or communication. The fields of a time slot of some processor p: (1) *task id* i and *instance id* j indicate the identifier of the time slot. (2) *start time st* and *finish time ft* indicate the start time and completion time of τ_i^j respectively. (3) *prev ptr* and *next ptr* are the pointers to the preceding and succeeding time slots respectively. The list is arranged in an increasing order of *start_time*. Any two time slots are nonoverlapping. Since the execution of an instance is nonpreemptable, the time difference between *start_time* and *finish_time* equals the execution time of the task.



Unscheduled Instance: ■ where $F < \tau < LCM$

Figure 3: Insertion of a new time slot

3.2.1 Recurrence

Given any solution point $(\phi, \sigma_m, \sigma_c)$, we construct the schedule by inserting time slots to the linked lists. Let σ_m: *task_id* × *instance_id* → *integer*. The insertion of a time slot for τ_i^j precedes that for τ_k^l if $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^l)$.

Recall that Equations 6 and 7 specify the bounds of the scheduling window for a task instance. Due to the communications, *est*(τ_i^j) in Equation 6 may not be the earliest time for τ_i^j. We define the *effective start time* as the time when (1) the hybrid constraints are satisfied and (2) τ_i^j receives all necessary data or messages from all the senders.

Given the effective start time τ and the assignment of τ_i (i.e. $p = \phi(\tau_i)$), a time slot of processor p is assigned to τ_i^j where *start_time* $\geq \tau$ and *finish_time* $-$ *start_time* $= e_j$. Note that we have to make sure the new time slot does not overlap existent time slots. Since (1) the executions of all instances within one scheduling frame recur in the next scheduling frame and (2) it is possible that the time slot for some instance is over LCM, we subtract one LCM from the *start_time* or *finish_time* if it is greater than LCM. It means the time slot for this task instance will be modulated

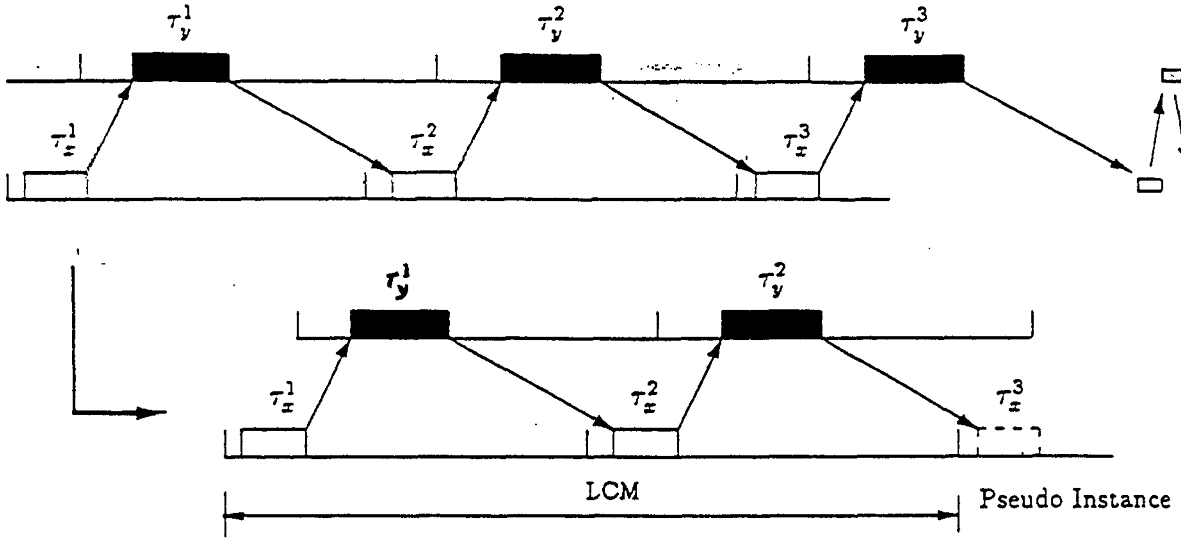


Figure 4: The introduction of a pseudo instance

and wrapped to the beginning of the schedule. As shown in Figure 3 The *start_time* of the new slot is τ while the completion time is $\tau + e$−LCM.

3.3 Pseudo Instances

As stated in Section 2, we consider the communication pattern in which cyclic dependency exists among tasks. Given a set of tasks, Γ, a set of task instances, I, a set of communications, C, and any solution point, $(\phi, \sigma_m, \sigma_c)$, we introduce pseudo instances to solve this problem. For any task τ_x, if there exists a task τ_y, in which (1) $\sigma_m(\tau_x^i) < \sigma_m(\tau_y^i)$, $\forall\, i$, (2) $n_x = n_y$, and (3) $\tau_x \mapsto \tau_y \in C$ and $\tau_y \mapsto \tau_x \in C$, then a pseudo instance $\tau_x^{n_x+1}$ is added to I. A pseudo instance is always a receiving instance. No insertion of time slots for pseudo instances is needed. For a pseudo instance, only the effective start time is concerned. The effective start time of a pseudo instance $\tau_x^{n_x+1}$ in the constructed schedule based on $(\phi, \sigma_m, \sigma_c)$ is checked to see whether it is less than LCM + s_x^1 or not. If yes, then the execution of τ_x^1 for the next scheduling frame may start at $LCM + s_x^1$ which is exactly one LCM away from the execution of τ_x^1 for the current scheduling frame. A graphical illustration of the introduction of pseudo instance to solve the synchronous communications of cyclic dependency is given in Figure 4 in which $n_x = 2$.

As for the asynchronous communications of cyclic dependency, no pseudo instances are needed. For example, if both $\tau_x \mapsto \tau_y$ and $\tau_y \mapsto \tau_x$ exist and $n_x = n_y \times n$, then for each τ_y^j, where $j = 1$, $2, \ldots, n_y$, find a sending instance $\tau_x^i \in I$ and a receiving instance $\tau_x^k \in I$ such that (1) $f_x^i \leq s_y^j$, (2) $f_y^j \leq s_x^k$, and (3) $\tau_x^i \mapsto \tau_y^j$ and $\tau_y^j \mapsto \tau_x^k$ are the communications. The relationship between i, j,

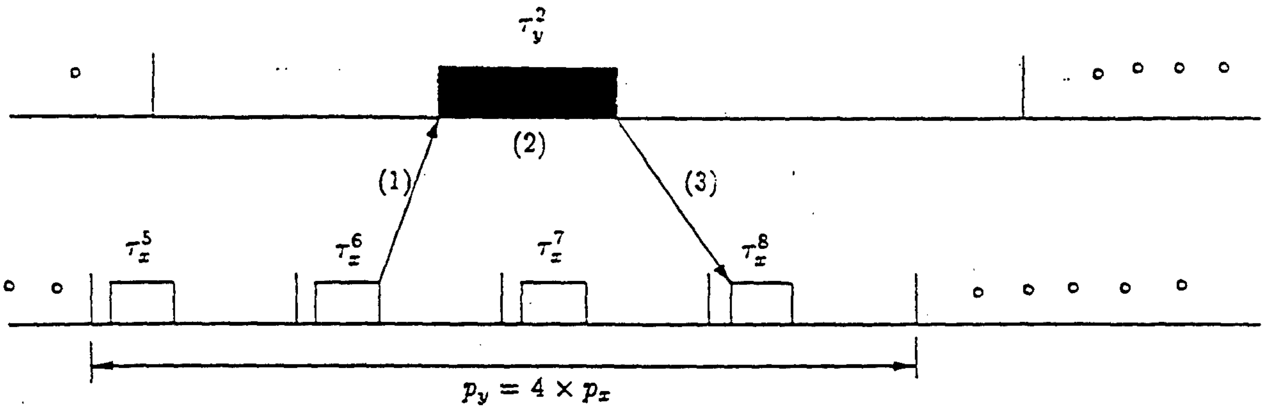


Figure 5: Asynchronous communications in mutuality

and k can be stated as

$$(j-1) \times n < i < k \leq j \times n. \tag{8}$$

A graphical illustration can be found in Figure 5. In the example, the values of i, j, k, and n are 6, 2, 8, 4 respectively. The communications $\tau_x^6 \mapsto \tau_y^2$ and $\tau_y^2 \mapsto \tau_x^8$ are scheduled before and after the scheduling of τ_y^2 respectively.

4 The Simulated Annealing Algorithm

Kirkpatrick *et al.* [KGV83] proposed a simulated annealing algorithm for combinatorial optimization problems. Simulated annealing is a global optimization technique. It is derived from the observation that an optimization problem can be identified with a fluid. There exists an analogy between finding an optimal solution of a combinatorial problem with many variables and the slow cooling of a molten metal until it reaches its low energy ground state. Hence, the terms about energy function, temperature, and thermal equilibrium are mostly used. During the search of an optimal solution, the algorithm always accepts the downward moves from the current solution point to the points of lower energy values, while there is still a small chance of accepting upward moves to the points of higher energy values. The probability of accepting an uphill move is a function of current temperature. The purpose of hill climbing is to escape from a local optimal configuration. If there are no upward or downward moves over a number of iterations, the thermal equilibrium is reached. The temperature then is reduced to a smaller value and the searching continues from the current solution point. The whole process terminates when either (1) the lowest energy point is found or (2) no upward or downward jumps have been taken for a number of successive thermal equilibrium.

The structure of simulated annealing (SA) algorithm is shown in Figure 7. The first step of the algorithm is to randomly choose an assignment ϕ, a total ordering of instances within one scheduling frame, σ_m, and a total ordering of communications for the instances, σ_c. A solution point in the search space of SA is a 3-tuple $(\phi, \sigma_m, \sigma_c)$. The energy of a solution point is computed by equation (5). For each solution point P which is infeasible, (i.e. E_p is nonzero), a neighbor finding strategy is invoked to generate a neighbor of P. As stated before, if the energy of the neighbor is lower than the current value, we accept the neighbor as the current solution; otherwise, a probability function (i.e. $exp(\frac{E_p - E_n}{T})$) is evaluated to determine whether to accept the neighbor or not. The parameter of the probability function is the current temperature. As the temperature is decreasing, the chance of accepting an uphill jump (i.e. a solution point with a higher energy level) is smaller. The inner and outer loops are for thermal equilibrium and termination respectively. The number of iterations for the inner loop is also a function of current temperature. The lower the temperature is, the bigger the number is. Methods about how to model the numbers of iterations and how to assign the number for each temperature have been proposed [LH91]. In this dissertation, we consider a simple incremental function. Namely, $N = N + \Delta$ where N is the number of iterations and Δ is a constant. The termination condition for the outer loop is $E_p = 0$. Whenever thermal equilibrium is reached at a temperature, the temperature is decreased. Linear or nonlinear approach of temperature decrease function can be simple or complex. Here we consider a simple multiplication function (i.e. $T = T \times \alpha$, where $\alpha < 1$).

4.1 Evaluation of Energy Value for a Solution Point $(\phi, \sigma_m, \sigma_c)$

The computation of the energy value stated in Equation 5 , is done by constructing multi-processor schedules and a network schedule, and collecting the the start and completion times of each task instance and communication from these schedules.

The construction of the schedules is characterized by the priority assignment of the task instances in the set. The priority assignment algorithm determines the scheduling order among all the task instances. Each time when a task instance is chosen to be scheduled, the incoming communications of the instance are scheduled first and then the task instance itself. After all the task instances have been scheduled, the scheduling of the outgoing communications is performed. An algorithmic description about how to compute the energy value for a solution point is given in Figure 6. Note that a communication is an incoming communication to a task instance if the frequency of the receiving task instance is equal to or less than that of the sending task instance. For example, $\tau_k^? \mapsto \tau_i^j$ and $\tau_k^j \mapsto \tau_i^j$ are incoming communications to τ_i^j. On the other hand, if the sender frequency is less than the receiver frequency, then the communication is an outgoing communication. (e.g. $\tau_k^j \mapsto \tau_i^?$ is the outgoing communication of τ_k^j).

4.1.1 Priority Assignment of Task Instances: σ_m

In the work [CA93], we presented the SLsF algorithm and the performance evaluation. The results showed that SLsF outperforms SPF and SJF. In this paper we use the SLsF as the priority assignment algorithm for the task instances in I.

Formally, if $lst(\tau_i^j) < lst(\tau_k^\ell)$, then $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$. And the insertion of a time slot for τ_i^j precedes that for τ_k^ℓ if $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$. The time-based scheduling algorithm for a task instance is used to find a time slot for a task instance once the *effective start time* is given. We define the *effective start time* of a task instance as the earliest start time when the incoming communications are taken into account. Let t be the maximum completion time among all the incoming communications of a task instance, then the *effective start time* of the task instance is set to the bigger value among t and est (as stated in Equation 6).

4.1.2 Scheduling the Incoming Communications:σ_c

There are two kinds of incoming communications. The first kind is called the synchronous communication in which the frequencies of the sender and receiver are identical. The other kind is called the asynchronous communication in which the sending task instance is associated with a question mark. For such an asynchronous communication, we have to decide which instance of the sending task should communicate with the receiving task instance. The approach we take is to find the nearest instance of the sending task. The reason is that, by finding the nearest instance, the time difference between start time of the receiving instance and the completion time of the sending instance is the smallest. The chance of violating the latency constraint of a communication will be the smallest then.

The nearest instance of a sending task can be found using the following method. Given an incoming communication $\tau_k^? \rightarrow \tau_i^j$, and the *effective start time* of τ_i^j, eft we search through the linked list of processor $\phi(\tau_k)$ up to time eft. If there is some instance of τ_k, say τ_k^ℓ, whose completion time is the latest among all scheduled instances of τ_k, then the nearest instance is found. Otherwise, we continue to search through the linked list until an instance of τ_k is found. We set the *effective start time* of the communication to be the completion time of the found instance. We also erase the question mark such that $\tau_k^? \rightarrow \tau_i^j$ is changed to $\tau_k^\ell \rightarrow \tau_i^j$. For the synchronous communication, the *effective start time* of the communication is simply assigned as the *finish time* of the sending task instance.

The scheduling of the communication is done by inserting a time slot to the linked list for the communications network. The *start time* of the time slot can not be earlier than the *effective start* *time* of the communication. Once the time slot is inserted, we check the *effective start time* of τ_i^j to make sure that it is not less than the *finish time* of the time slot. If it is, the *effective start time* of τ_i^j is updated to be the *finish time* of the time slot.

If a task instance has more than one incoming communication, the scheduling order among these communications is based on their latency constraints. The bigger the latency value is, the earlier the communication is scheduled. The incoming communication with the tightest latency constraint is scheduled last. It is because the *effective start time* of the receiving task instance is constantly updated by the scheduling of the incoming communications. It is possible that the scheduling of the later incoming communications increases the *effective start time* of the receiving task instance and make the early scheduled communication violate its latency constraint if the constraint is tight.

4.1.3 Scheduling the Outgoing Communications: σ_c

The scheduling of the outgoing communications for the whole task set is performed after all the task instances have been scheduled. The scheduling order among these communications is based on the finish times of the sending task instances. The task instance with the smallest finish time is considered first. When a task instance is taken into account, all its outgoing communications are scheduled one by one according to their latency constraints. The communication with the tightest latency constraint is scheduled first.

Given an outgoing communication $\tau_i^j \leftarrow \tau_k^?$, and the finish time of τ_i^j, f_i^j, the *effective start time* of the communication is set to be f_i^j. Based on the *effective start time*, a time slot in inserted for this communication. Then the nearest instance of receiving task can be found based on the *finish time* of the time slot.

For the example shown in Figure 5, The incoming communication marked with "(1)" is scheduled before the scheduling of τ_y^2. The sixth instance of τ_z is chosen as the nearest instance. As for the outgoing communication marked with "(3)", it is scheduled after the scheduling of τ_z^5, τ_z^6, τ_z^7, and τ_z^8. In this example, τ_z^8 is the nearest instance of the outgoing communication.

4.2 Neighbor Finding Strategy: ϕ

The neighbor finding strategy is used to find the next solution point once the current solution point is evaluated as infeasible (i.e. energy value is nonnegative). The neighbor space of a solution point is the set of points which can be reached by changing the assignment of one or two tasks. There are several modes of neighbor finding strategy.

- Balance Mode: We randomly move a task from the heavily-loaded processor to the lightest-loaded processor. This move tries to balance the workload of processors. By balancing the workload, the chance to find a neighbor with a lower energy value is bigger.

- Swap Mode: We randomly choose two tasks τ_i and τ_j on processors p and q respectively. Then we change ϕ by setting $\phi(\tau_i) = q$ and $\phi(\tau_j) = p$.

- Merge Mode: We pick two tasks and move them to one processor. By merging two tasks to a processor, we increase the workload of the processor. There is an opportunity of increasing the energy level of the new point by increasing the workload of the processor. The purpose of the move is to perturb the system and allow the next move to escape from the local optimum.

- Direct Mode: When the system is in a low-energy state, only few tasks violate the jitter or latency constraints. Under such a circumstance, it will be more beneficial to change the assignment of these tasks instead of randomly moving other tasks. From the conducted experiments, we find that this mode can accelerate the searching of a feasible solution especially when the system is about to reach the equilibrium.

The selection of the appropriate mode to find a neighbor is based on the current system state. Given a randomly generated initial state (i.e. solution point), the workload discrepancy between the processors may be huge. Hence, in the early stage of the simulated annealing, the balance mode is useful to balance the workload. After the processor workload is balanced out, the swap mode and the merge mode are frequently used to find a lower energy state until the system reaches near-termination state. In the final stage of the annealing, the direct mode tries to find a feasible solution. The whole process terminates when a feasible solution is found in which the energy value is zero.

5 Experimental Results

We implemented the algorithm as the framework of the allocator on *MARUTI* [GMK+91, MSA92, SdSA94], a real-time operating system developed at the University of Maryland, and conducted extensive experiments under various task characteristics. The tests involve the allocation of real-time tasks on a homogeneous distributed system connected by a communication channel.

To test the practicality of the approach and show the significance of the algorithm, we consider a simplified and sanitized version of a real problem. This was derived from actual development work, and is therefore representative of the scheduling requirements of an actual avionics system. The Boeing 777 Aircraft Information Management System (AIMS) is to be running on a multiprocessor

	10_Proc	9_Proc	8_Proc	7_Proc	6_Proc
Exec_Time (Sec)	2369	5572	19774	36218	78647
$= Hr : Min : Sec$	0:39:29	1:32:52	5:29:34	10:03:38	21:50:47

Table 1: The execution times of the AIMS with different number of processors

system connected by a SafeBus (TM) ultra-reliable bus. The problem is to find the minimum number of processors needed to assign the tasks to these processors. The objective is to develop an off-line non-preemptable schedule for each processor and one schedule for the SafeBus (TM) ultra-reliable bus.

The AIMS consists of 155 tasks and 951 communications between these tasks. The frequencies of the tasks vary from 5HZ to 40HZ. The execution times of the tasks vary from $0ms$ to $16.650ms$. The NEI and XEI of a task t_i are $p_i - 500\mu s$ and $p_i + 500\mu s$ respectively. Since $\delta = 1000\mu s = 1ms$ $< \frac{25ms}{e}$, the smallest-period-first scheduling algorithm can be used in this case. Tasks communicate with others asynchronously and in mutuality. The transmission times for communications are in the range from $0\mu s$ to $447.733\mu s$. The latency constraints of the communications vary from $68.993ms$ to $200ms$. The LCM of these 155 tasks is $200ms$. When the whole system is extended, the total number of task instances within one scheduling frame is 624 and the number of communications is 1580.

For such a real and tremendous problem size, pre-analysis is necessary. We calculate the resource utilization index to estimate the minimum number of processors needed to run AIMS. The index is defined as

$$\frac{\sum_{i=1}^{155}(e_i \times g_i)}{LCM}$$

where e_i is the execution of task t_i and $g_i = \frac{LCM}{p_i}$. The obtained index for AIMS is 5.14. It means there exist no feasible solutions for the AIMS if the number of processors in the multiprocessor system is less than 6.

The number of processors which the AIMS is allowed to run on is a parameter to the scheduling problem. We start the AIMS scheduling problem with 10 processors. After a feasible solution is found, we decrease the number of processors by one and solve the whole problem again. We run the algorithm on a DECstation 5000. The execution time for the AIMS scheduling problem with different numbers of processors is summarized in Table 1. The algorithm is able to find a feasible solution of the AIMS with six processors which is the minimum number of processors according to the resource utilization index. The time to find such a feasible solution is less than one day (approximately 22 hours).

5.1 Discussions

For feasible solutions of the AIMS with various numbers of processors, we calculate the processor utilization ratio (PUR) of each processor. The processor utilization ratio for a processor p is defined as

$$\frac{\sum_{\phi(t_i)=p}(e_i \times q_i)}{LCM}.$$

The results are shown in Figure 8. The ratios are sorted into a non-decreasing order given a fixed number of processors. The algorithm generates the feasible solutions for the AIMS with 6, 7, 8, 9 and 10 processors respectively. For example, for the 6-processor case, the PURs for the heaviest-loaded and lightest-loaded processors are 0.91 and 0.76 respectively. For the 10-processor cases, the PURs are 0.63 and 0.28 respectively. We find that the ratio difference between the heaviest-loaded processor and the lightest-loaded processor in the 6-processor case is smaller than those in other cases. It means the chance for a more load-balanced allocation to find a feasible solution is bigger when the number of processors is smaller.

The detailed schedules for the 6-processor case are shown in Figure 9. The results are shown on an interactive graphical interface which is developed for the design of *MARUTI*. The time scale shown in Figure 9 is $100\mu s$. So the LCM is shown as 2000 in the figure. (i.e. $2000 \times 100\mu s = 200ms$.) This solution consists of seven off-line non-preemptive schedules: one for each processor and one for the SafeBus (TM). Each of these schedules will be one LCM long where an infinite schedule can be produced by repeating these schedules indefinitely. Note that the pseudo instances are introduced to make sure the wrapping around at the end of the LCM-long schedules should satisfy the latency and next-execution-interval requirements across the point of wrap-around. The pseudo instances are not shown in Figure 9.

The inclusion of resource and memory constraints into the problem can be done by modifying neighbor-finding strategy. Once a neighbor of the current point is generated, it is checked to ascertain that the constraints on memory etc. are met. If not, the neighbor is discarded and another neighbor is evaluated.

References

[CA93] Sheng-Tzong Cheng and Ashok K. Agrawala. Scheduling of periodic tasks with relative timing constraints. Technical Report CS-TR-3392, UMIACS-TR-94-135, Department of Computer Science, University of Maryland, College Park, December. 1993. Submitted to *the 10th Annual IEEE Conference on Computer Assurance, COMPASS '95.*

[CDHC94] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. Arinc 659 scheduling: Problem definition. In *Proceedings of IEEE Real-Time Systems Symposium*, San Juan, PR, Dec. 1994.

[GMK+91] Ó. Gudmundsson, D. Mossé, K.T. Ko, A.K. Agrawala, and S.K. Tripathi. Maruti: A platform for hard real-time applications. In K. Gordon, A.K. Agrawala, and P. Hwang (eds.), editors, *Mission Critical Operating Systems.* IOS Press, 1991.

[HS92] Chao-Ju Hou and Kang G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *Proceedings of the 1992 IEEE 13th Real-Time Systems Symposium*, pages 146–155, Phoenix, AZ, 1992.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[LH91] Feng-Tse Lin and Ching-Chi Hsu. Task assignment problems in distributed computing systems by simulated annealing. *Journal of the Chinese Institute of Engineers*, 14(5):537–550, Sept. 1991.

[MSA92] Daniel Mossé, M.C. Saksena, and Ashok K. Agrawala. Maruti: An approach to real-time system design. Technical Report CS-TR-2845, UMIACS-TR-92-21, Department of Computer Science, University of Maryland, College Park, 1992.

[Ram90] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 108–115, Paris, France, 1990.

[SdSA94] M. Saksena, J. da Silva, and A. K. Agrawala. Design and implementation of *maruti-ii.* Technical Report CS-TR-2845, Department of Computer Science, University of Maryland, College Park, 1994.

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.

```
Given a solution point $P = (\phi, \sigma_m, \sigma_c)$
While there is some unscheduled task instance do
        Find the next unscheduled instance. /* By the SLsF algorithm */
        Let the instance be $\tau_i^j$.
        Sort all the incoming communications of $\tau_i^j$ based on
                the latency values into a descending order.
        Schedule each incoming communication starting from
                the biggest-latency one to the tightest-latency one.
        Schedule the instance $\tau_i^j$.
End While.
Mark each instance as un-examined.
While there is some un-examined task instance do
        Find the next un-examined task instance. /* By the finish times */
        Sort all the outgoing communications of the task instance based
                on the latency values into an increasing order.
        Schedule each outgoing communication starting from
                the tightest-latency one to the biggest-latency one.
        Mark the task instance examined.
End While.
Collect the start time and finish time informations for each task instance and communication.
Compute the energy value using Equation 5.
```

Figure 6: The pseudo code for computing the energy value

```
Choose an initial temperature $T$
Choose randomly a starting point $P = (\phi, \sigma_m, \sigma_c)$
$E_p$ := Energy of solution point $P$
if $E_p = 0$ then
    output $E_p$ and exit /* $E_p = 0$ means a feasible solution */
end if
repeat
    repeat
        Choose $N$, a neighbor of $P$
        $E_n$ := Energy of solution point $N$
        if $E_n = 0$ then
            output $E_n$ and exit /* $E_n = 0$ means a feasible solution */
        end if
        if $E_n < E_p$ then
            $P := N$
            $E_p := E_n$
        else
            $x := \frac{E_p - E_n}{T}$
            if $e^x \geq$ random(0,1) then
                $P := N$
                $E_p := E_n$
            end if
        end if
    until *thermal equilibrium at* $T$
    $T := \alpha \times T$ (where $\alpha < 1$)
until *stopping criterion*
```

Figure 7: The structure of simulated annealing algorithm.

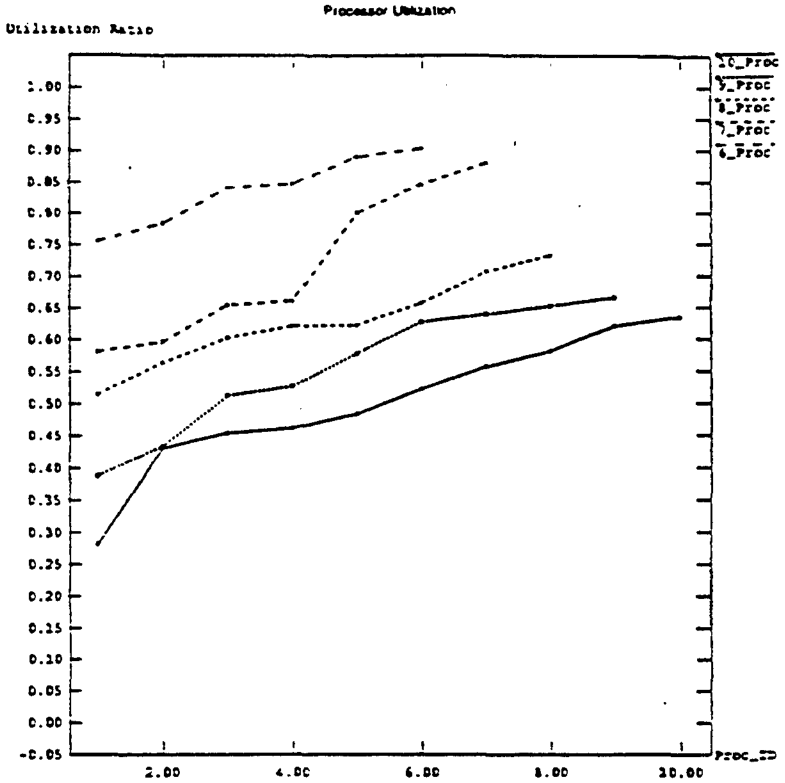
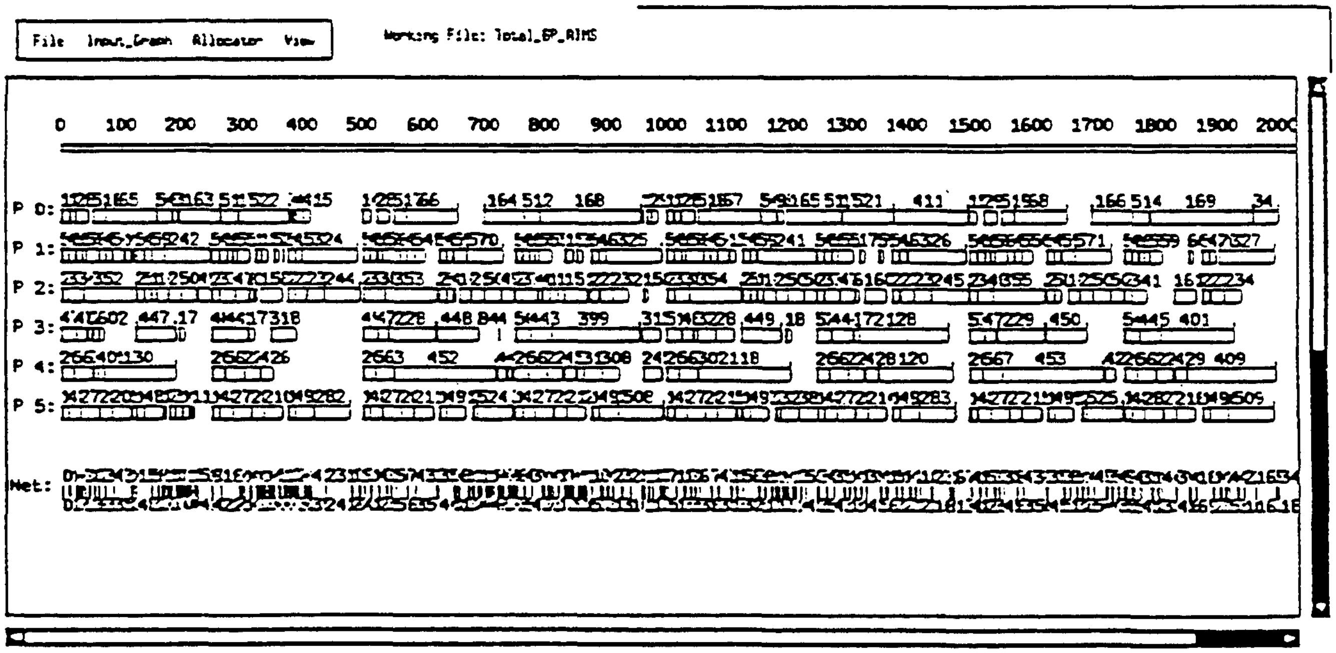


Figure 8: Processor Utilization Ratios for different cases



Working File: Total_EP_AIMS

Figure 9: The Allocation Results and Schedules for AIMS with 6 processors

REPORT DOCUMENTATION PAGE

Form approved
OMB No 074-0188

1. **AGENCY USE ONLY (leave blank)**

2. **REPORT DATE**
January 1995

3. **REPORT TYPE END DATES COVERED**
Technical Report

4. **TITLE AND SUBTITLE**
Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints

5. **FUNDING NUMNBERS**
N00014091-C-0195
DSAG-60-C-0055

6. **AUTHOR(S)**
Sheng-Tzong Cheng and Ashok K. Agrawala

7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
University of Maryland
Department of Computer Science
A.V. Williams Building
College Park, MD 20742

8. **PERFORMING ORGANIZATION REPORT NUMBER**
CS-TR-3402
UMIACS-TR-95-6

9. **SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Honeywell Inc.
3600 Technology Drive
Minneapolis, MN 55148

Phillips Laboratory
Directorate of Contracting
3651 Lowry Avenue, SE
Kirtland AFB, NM 87117-5777

10. **SPONSORING/ MONITORING AGENCY REPORT NUMBER**

11. **SUPPLEMENTARY NOTES**

12.a. **DISTRIBUTION/ AVAILABILITY STATEMENT**

12.b. **DISTRIBUTION CODE**

13. **ABSTRACT** (Maximum 200 words)

Allocation problem has always been one of the fundamental issues of building the applications in distributed computing systems (DCS). For real-time applications on DCS, the allocation problem should directly address the issues of task and communication scheduling. In this context, the allocation of tasks has to fully utilize the available processors and the scheduling of tasks has to meet the specified timing constraints. Clearly, the execution of tasks under the allocation and schedule has to satisfy the precedence, resources, and other synchronization constraints among them.

Recently, the timing requirements of the real-time systems emerge that the relative timng constraints are imposed on the consecutive executions of each task and the inter-task temporal relationships are specified across task periods. In this paper we consider the allocation and scheduling problem of the periodic tasks with such timing requirements. Given a set of periodic tasks, we consider the least common multiple (LCM) of the task periods. Each task is extended to several instances within the LCM. The scheduling window for each task instance is derived to satisfy the timing constraints. We develop a simulated annealing algorithm as the overall control algorithm. An example problem of the sanitized version of the Boeing 777 Aircraft Information Management Systems is solved by the algorithm. Experimental results show that the algorithm solves the problem in a reasonable time complexity.

14. **SUBJECT TERMS**
Process Management; Special Purpose and Application-Based Systems

15. **NUMBER OF PAGES**
22

16. **PRICE CODE**

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlisted

Scheduling of Periodic Tasks with Relative Timing Constraints *

Sheng-Tzong Cheng and Ashok K. Agrawala
Institute for Advanced Computer Studies
Systems Design and Analysis Group
Department of Computer Science
University of Maryland
College Park, MD 20742
{stcheng,agrawala}@cs.umd.edu

Abstract

The problem of non-preemptive scheduling of a set of periodic tasks on a single processor has been traditionally considering the ready time and deadline on each task. As a consequence, a feasible schedule finds that in each period one instance of each task starts the execution after the ready time and completes the execution before the deadline .

Recently, the timing requirements of the real-time systems emerge that the relative timing constraints are imposed on the consecutive executions of each task. In this paper, we consider the scheduling problem of the periodic tasks with the relative timing constraints imposed on two consecutive executions of a task. We analyze the timing constraints and derive the scheduling window for each task instance. Based on the scheduling window, we present the time-based approach of scheduling a task instance. The task instances are scheduled one by one based on their priorities assigned by the proposed algorithms in this paper. We conduct the experiments to compare the schedulability of the algorithms.

*This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies. either expressed or implied. of Honeywell or Army/Phillips.

1 Introduction

The task scheduling problem is one of the basic issues of building real-time applications in which the tasks of applications are associated with timing constraints. For the hard real-time applications, such as avionics systems and nuclear power systems, the approach to guarantee the critical timing constraints is to schedule periodic tasks *a priori*. A non-preemptive schedule for a set of periodic tasks is generated by assigning a start time to each execution of a task to meet their timing constraints. Failure to meet the specified timing constraints can result in disastrous consequence.

Various kinds of periodic task models have been proposed to represent the real-time system characteristics. One of them is to model an application as a set of tasks, in which each task is executed once every period under the ready time and deadline constraints. These constraints impose constant intervals in which a task can be executed. In literature, many techniques [2, 3, 4, 5, 6, 7, 8] have been proposed to solve the scheduling problem in this context. The deficiency of this modeling is the inability of specifying the relative constraints across task periods. For example, one can not specify the timing relationship between two consecutive executions of the same task.

Simply assuring that one instance of each task starts the execution after the ready time and completes the execution before the specified deadline is not enough. Some real-time applications have more complicated timing constraints for the tasks. For example, the relative timing constraints may be imposed upon the consecutive executions of a task in which the scheduling of two consecutive executions of a periodic task must be separated by a minimum execution interval. The Boeing 777 Aircraft Information Management System is such an example [1]. One possible solution to the scheduling problem of such applications is to consider the instances of tasks rather than the tasks. A task instance is defined as one execution of a task within a period. With the notion of task instances, one is able to specify the various timing constraints and dependencies among instances of tasks.

In this paper, we consider the relative timing constraints imposed on two consecutive instances of a task. The task model and the analysis of the timing constraints are introduced in Sections 2 and 3 respectively. Based on the analysis, we are able to derive the scheduling window for each task instance. Given the scheduling window of a task instance, we present the time-based approach of scheduling a task instance in Section 4. We propose three priority assignment algorithms for the task instances in Section 5. The task instances are scheduled one by one based on their priorities. In Section 6, we evaluate the three algorithms and show the experimental results.

2 Problem Statement

Consider a set of periodic tasks $\Gamma = \{ \tau_i \mid i = 1, \dots n \}$, where τ_i is a 4-tuple $< p_i, e_i, \lambda_i, \eta_i >$ denoting the period, computation time, low jitter and high jitter respectively. One instance of a task is executed each period. The execution of a task instance is non-preemptable. The start times of two consecutive instances of task τ_i are at least $p_i - \lambda_i$ and at most $p_i + \eta_i$ apart.

In order to schedule periodic tasks, we consider the least common multiple (LCM) of all periods of tasks. Let n_i be the number of instances for task τ_i within a schedule of length LCM. Hence, $n_i = \frac{\mathrm{LCM}}{p_i}$. A schedule for a set of tasks is the mapping of each task τ_i to n_i task instances and the assigning of a start time s_i^j to the j-th instance of task τ_i, τ_i^j, $\forall\ i = 1, \dots n$ and $j = 1, \dots, n_i$. A *feasible* schedule is a schedule in which the following conditions are satisfied for each task τ_i:

$$\begin{aligned} f_i^j &= s_i^j + e_i && (1) \\ s_i^{n_i+1} &= s_i^1 + \mathrm{LCM} && (2) \\ s_i^j &\geq s_i^{j-1} + p_i - \lambda_i && (3) \\ s_i^j &\leq s_i^{j-1} + p_i + \eta_i && (4) \\ & \forall j = 2, \dots, n_i + 1. \end{aligned}$$

The non-preemption scheduling discipline leads to Equation 1 where f_i^j is the finish time of τ_i^j. Another condition for non-preemption scheduling is that given any i, j, k and ℓ, if $s_i^j < s_k^\ell$ then $f_i^j \leq s_k^\ell$. It means the schedule for any two instances is non-overlapping. The constructed schedule of length LCM is invoked repeatedly by wrapping-around the end point of the first schedule to the start point of the next one. Hence, as shown in Equation 2, the start time of the first instance in the next schedule is exactly one LCM away from that of the first schedule. Finally, Equations 3 and 4 specify the relative timing constraints between two consecutive instances of a task.

3 Analysis of Relative Timing Constraints

Define the scheduling window for a task instance as the time interval during which the task can start. Traditionally, the lower and upper bounds of the scheduling window for a task instance are called earliest start time (*est*) and latest start time (*lst*) respectively. These values are given and independent of the start times of the preceding instances.

Instance ID	$est = s_i^{j-1} + p_i - \lambda_i$	$lst = s_i^{j-1} + p_i + \eta_i$	actual start time (s_i^j)
τ_i^1	0	40	4
τ_i^2	39	49	40
τ_i^3	75	85	77
τ_i^4	112	122	113
τ_i^5	148	158	*

Table 1: An example to show the wrong setting of scheduling windows

We consider the scheduling of periodic tasks with relative timing constraints described in Equations 3 and 4. The scheduling window for a task instance is derived from the start times of its preceding instances. A *feasible* scheduling window for a task instance τ_i^j is a scheduling window in which any start time in the window makes the timing relation between s_i^{j-1} and s_i^j satisfy Equations 3 and 4. Formally, given s_i^1, s_i^2, ..., and ..., s_i^{j-1}, the problem is to derive the feasible scheduling window for τ_i^j such that a feasible schedule can be obtained if τ_i^j is scheduled within the window.

For the sake of simplicity, we assume that $r_i = 0$ and $d_i = p_i$, $\forall\, i$, in this section. Then, simply assigning *est* and *lst* of τ_i^j as $s_i^{j-1} + p_i - \lambda_i$ and $s_i^{j-1} + p_i + \eta_i$ respectively where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n_i$, is not tight enough to guarantee a feasible solution. For example, consider the case shown in Table 1 in which a periodic task τ_i is to be scheduled. Let LCM, p_i, λ_i, and η_i be 200, 40, 5, and 5 respectively. Hence, there are 5 instances within one LCM (i.e. $n_i = 5$). The first column in Table 1 indicates the instance IDs. The second and third columns give the *est* and *lst* of the scheduling windows for the task instances specified in the first column. The last column shows the actual start times scheduled for the particular task instances. The actual start time is a value in between *est* and *lst* of each task instance. For instance, the *est* and *lst* of τ_i^2 are 39 and 49 respectively. It means $39 \le s_i^2 \le 49$. The scheduled value for s_i^2, in the example, is 40. Since $s_i^6 = s_i^1 + \text{LCM} = 204$, we find that any value in the interval [148,158] can not satisfy the relative timing constraints between τ_i^5 and τ_i^6. As a consequence, the constructed schedule is infeasible.

We draw a picture to depict the relations among the start times of task instances in Figure 1. When τ_i^j is taken into account, the scheduling window for s_i^j is obtained by considering its relation with s_i^{j-1} as well as that with $s_i^{n_i}$ and $s_i^{n_i+1}$. We make sure that once s_i^j is determined, the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible scheduling window for $s_i^{n_i}$. Namely, the interval which is specified by the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j, overlaps the interval

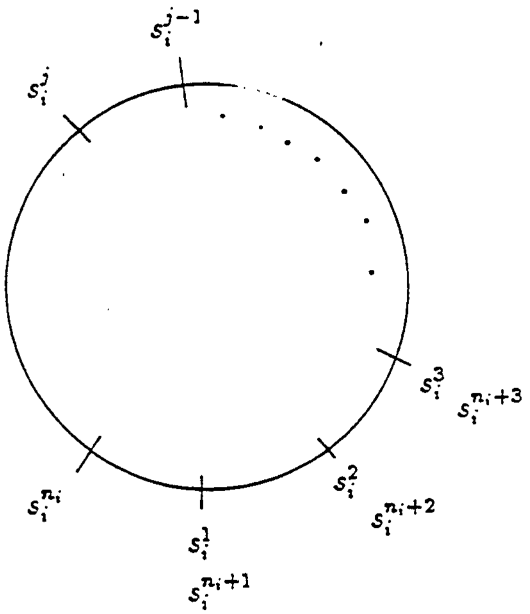


Figure 1: The relations between the task instances

$[s_i^{n_i+1} - (p_i + \eta_i),\ s_i^{n_i+1} - (p_i - \lambda_i)]$.

Proposition 1: Let the est and lst of τ_i^j be

$$est(\tau_i^j) = max\{(s_i^{j-1} + p_i - \lambda_i),\ (s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)\}, \qquad (5)$$

$$\text{and } lst(\tau_i^j) = min\{(s_i^{j-1} + p_i + \eta_i),\ (s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)\}. \qquad (6)$$

If s_i^j is in between the $est(\tau_i^j)$ and $lst(\tau_i^j)$, then the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible window.

Proof: Let ℓ and μ be the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j, respectively.

Hence,

$$\ell = s_i^j + (n_i - j) \times (p_i - \lambda_i) \qquad (7)$$

$$\mu = s_i^j + (n_i - j) \times (p_i + \eta_i) \qquad (8)$$

To guarantee the existence of feasible start time of $\tau_i^{n_i}$, the interval $[\ell,\mu]$ has to overlap the interval $[s_i^{n_i+1} - (p_i + \eta_i),\ s_i^{n_i+1} - (p_i - \lambda_i)]$. Hence the following conditions have to be satisfied:

$$s_i^{n_i+1} - \ell \geq p_i - \lambda_i \qquad (9)$$

$$s_i^{n_i+1} - \mu \le p_i + \eta_i \tag{10}$$

By replacing ℓ in Equation 9 with $s_i^j + (n_i - j) \times (p_i - \lambda_i)$, we obtain

$$\begin{aligned} s_i^j &\le s_i^{n_i+1} - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + \mathrm{LCM} - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + n_j \times p_i - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + (j - 1) \times p_i + (n_i - j + 1) \times \lambda_i \end{aligned} \tag{11}$$

Likewise, by replacing μ in Equation 10 with $s_i^j + (n_i - j) \times (p_i + \eta_i)$, we have

$$\begin{aligned} s_i^j &\ge s_i^{n_i+1} - (n_i - j + 1) \times (p_i + \eta_i) \\ &= s_i^1 + \mathrm{LCM} - (n_i - j + 1) \times (p_i + \eta_i) \\ &= s_i^1 + (j - 1) \times p_i - (n_i - j + 1) \times \eta_i \end{aligned} \tag{12}$$

So, According to Equations 12 and 3, we choose the bigger value between $(s_i^{j-1} + p_i - \lambda_i)$ and $(s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)$ as the *est* of τ_i^j. Similarly, according to Equations 11 and 4, we assign the smaller value of $(s_i^{j-1} + p_i + \eta_i)$ and $(s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)$ as the *lst*.

□

Example 3.1: To show how Proposition 3 gives a tighter bound to find feasible scheduling windows, we consider the case shown in Table 1 again. We apply Equations 5 and 6 to compute the *est* and *lst* of each instance. The results are shown in Table 2. Note that the scheduling windows for τ_i^4 and τ_i^5 are tighter than those in Table 1. As a consequence, any start time in the interval [159,160] for τ_i^5 satisfys the relative timing constraints between τ_i^5 and τ_i^6.

3.1 Property of Scheduling Windows

Define $P_i(x, y, z)$ as the predicate in which the estimated *est* and *lst* of τ_i^y, based on s_i^x and s_i^z, specify a feasible scheduling window for τ_i^y. In Proposition 3 , we prove that for any s_i^j in between *est*(τ_i^j) and *lst*(τ_i^j) as specified in Equations 5 and 6, $P_i(j, n_i, n_i + 1)$ is true.

Instance ID	*est* from Equation 5	*lst* from Equation 6	actual start time (s_i^j)
τ_i^1	0	40	4
τ_i^2	39	49	40
τ_i^3	75	85	77
τ_i^4	114	122	115
τ_i^5	159	160	$159 \sim 160$

Table 2: The correct setting of scheduling windows based on Proposition 3.1.

Lemma 1 *Given s_i^1, s_i^2, ..., and s_i^j, if, $\forall\ k = 2, \ldots, j$, $\mathrm{est}(\tau_i^k) \leq s_i^k \leq \mathrm{lst}(\tau_i^k)$ as specified in Equations 5 and 6, then $P_i(j, y, n_i + 1)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.*

Proof: We prove that the estimated *est* and *lst* of τ_i^y, based on s_i^j and $s_i^{n_i+1}$, specify a feasible scheduling window, by showing that (1) the estimated scheduling window of s_i^y, based on s_i^j, is specified by the interval

$$[s_i^j + (y-j) \times (p_i - \lambda_i), s_i^j + (y-j) \times (p_i + \eta_i)], \tag{13}$$

(2) the estimated scheduling window of s_i^y, based on $s_i^{n_i+1}$, is specified by the interval

$$[s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i), s_i^{n_i+1} - (n_i - y + 1) \times (p_i - \lambda_i)], \tag{14}$$

and (3) the intervals in Equations 13 and 14 overlap.

In Figure 2, we see that the necessary and sufficient conditions for the overlapping of the intervals specified in Equations 13 and 14 are

$$\begin{aligned} s_i^j + (y-j) \times (p_i - \lambda_i) &\leq s_i^{n_i+1} - (n_i - y + 1) \times (p_i - \lambda_i) && (15) \\ \text{and}\quad s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i) &\leq s_i^j + (y-j) \times (p_i + \eta_i). && (16) \end{aligned}$$

By solving the Equations 15 and 16, we obtain

$$\begin{aligned} s_i^j &\leq s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \\ \text{and}\quad s_i^j &\geq s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i. \end{aligned}$$

The above two equations describe the same conditions as Equations 11 and 12 do. Hence, $P_i(j, y, n_i + 1)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.

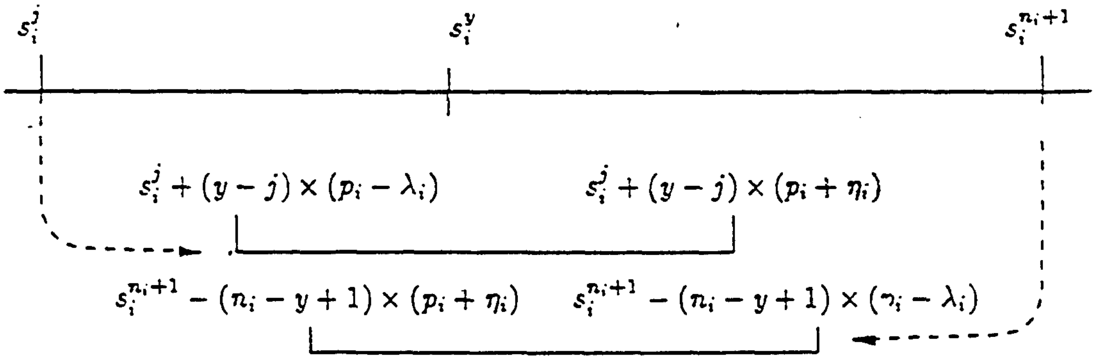

s_i^j
s_i^y
$s_i^{n_i+1}$
$s_i^j + (y - j) \times (p_i - \lambda_i)$
$s_i^j + (y - j) \times (p_i + \eta_i)$
$s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i)$
$s_i^{n_i+1} - (n_i - y + 1) \times (\eta_i - \lambda_i)$

Figure 2: The overlapping of two intervals

□

Lemma 2 *Given s_i^1, s_i^2, ..., s_i^j, and an integer n_0, where $1 \le n_0 \le j$, if, $\forall$ $k = 2$, ..., j, $est(\tau_i^k) \le s_i^k \le lst(\tau_i^k)$ are specified as in Equations 5 and 6, then $P_i(j, y, n_i + n_0)$ is true, $\forall$ $y = j + 1$, $j + 2$, ..., n_i.*

Proof: We use the same method in Lemma 1 to prove it. We show that (1) the estimated scheduling window of s_i^y, based on s_i^j, is specified by the interval

$$[s_i^j + (y - j) \times (p_i - \lambda_i), s_i^j + (y - j) \times (p_i + \eta_i)], \qquad (17)$$

(2) the estimated scheduling window of s_i^y, based on $s_i^{n_i+n_0}$, is specified by the interval

$$[s_i^{n_i+n_0} - (n_i + n_0 - y) \times (p_i + \eta_i), s_i^{n_i+n_0} - (n_i + n_0 - y) \times (p_i - \lambda_i)], \qquad (18)$$

and (3) these two intervals overlap.

The following conditions have to be satisfied to make sure the overlapping of the two intervals.

$$s_i^j \le s_i^{n_0} + (j - 1) \times p_i + (n_i - j + 1) \times \lambda_i - (p_i - \lambda) \times n_0 - 1 \qquad (19)$$

$$\text{and } s_i^j \ge s_i^{n_0} + (j - 1) \times p_i - (n_i - j + 1) \times \eta_i - (p_i + \eta_i) \times n_0 - 1. \qquad (20)$$

Since $s_i^1 \le s_i^{n_0} - (p_i - \lambda) \times (n_0 - 1)$ and $s_i^1 \ge s_i^{n_0} - (p_i + \eta_i) \times (n_0 - 1)$, we rewrite Equations 19 and 20

$$s_i^j \le \underline{s_i^{n_0}} + (j - 1) \times p_i + (n_i - j + 1) \times \lambda_i \underline{-(p_i - \lambda) \times n_0 - 1}$$

$$
\begin{aligned}
& \leq \underline{s_i^1} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \\
\text{and } s_i^j & \geq \underline{s_i^{n_0}} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i \underline{-(p_i + \eta_i) \times n_0 - 1}. \\
& \geq \underline{s_i^1} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i
\end{aligned}
$$

Hence $P_i(j, y, n_i + n_0)$ holds for any $1 \leq n_0 \leq j$.

□

Theorem 1 *Given* s_i^1, s_i^2, ..., *and* s_i^j, *if,* $\forall\ k = 2, \ldots, j$, $\text{est}(\tau_i^k) \leq s_i^k \leq \text{lst}(\tau_i^k)$ *as specified in Equations 5 and 6, then* $P_i(j, y, z)$ *is true,* $\forall\ y = j+1, j+2, \ldots, n_i$, *and* $z = n_i + 1, n_i + 2, \ldots, n_i + j$.

By combining the proofs in Lemmas 1 and 2, it is easy to see that Theorem 1 holds. Based on Theorem 1 , we can assign the scheduling window for τ_i^j by using Equations 5 and 6 once s_i^1, s_i^2, ..., s_i^{j-1}.

Before we present the scheduling technique for a task instance, let us consider the following objective. The objective can be formulated as follows. Given a set of tasks with the characteristics described in Section 2, we schedule the task instances for each task within one LCM to minimize

$$\pi = \sum_{i,j} \alpha(s_i^j - s_i^{j-1} - p_i) \qquad (21)$$

Subject to the constraints specified in Equations 1 through 4,

where $\alpha(x) = x$, if $x \geq 0$; $= -x$, otherwise.

Basically, we try to schedule every instance of a task one period apart from its preceding instance. An optimal schedule is a feasible schedule with the minimum total deviation value from one period apart for instances.

4 The Time-Based Scheduling of a Task Instance

We consider the time-based solution to the scheduling problem by using a linked list. Each element in the list represents a time slot assigned to a task instance. A time slot w has the following fields: (1) *task id* i and *instance id* j indicate the identifier of the time slot. (2) *start time st* and *finish time ft* indicate the start time and completion time of τ_i^j respectively. (3) *prev ptr* and *next ptr* are the

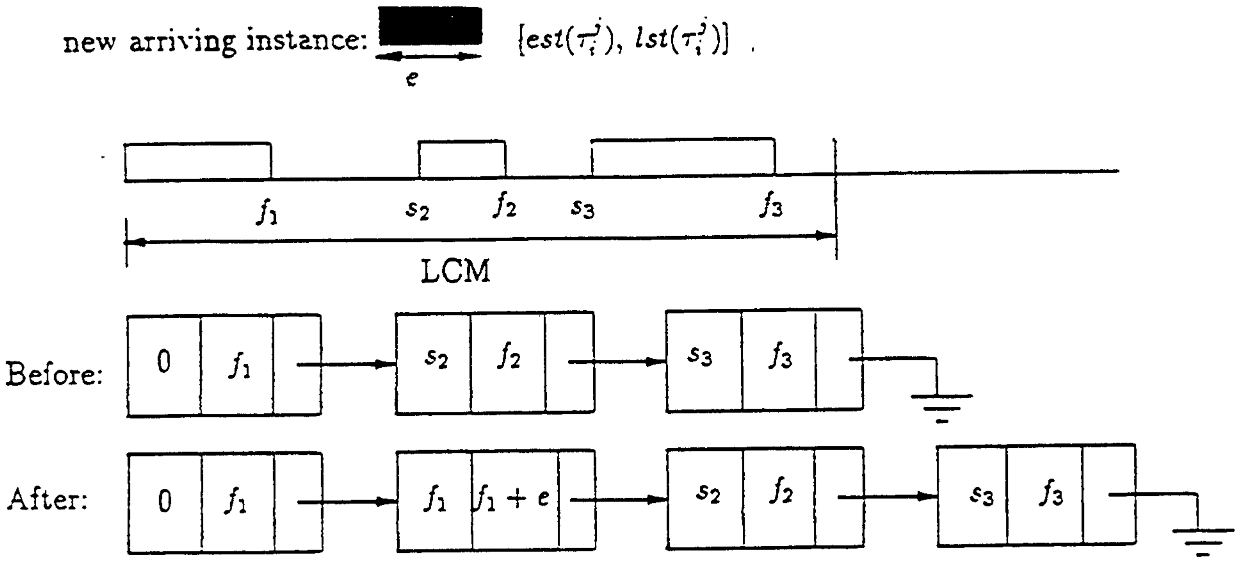


Figure 3: Insertion of a new time slot

pointers to the preceding and succeeding time slots respectively. We arrange the time slots in the list in increasing order by using the *start time* as the key. Any two time slots are non-overlapping. Since the execution of an instance is non-preemptable, the time difference between *start time* and *finish time* equals the execution time of the task.

4.1 Creating a Time Slot for the Task Instance

Consider a set of n tasks. Given a linked list and a task instance τ_i^j, we schedule the instance by inserting a time slot to the list. According to equations 5 and 6, we compute the $est(\tau_i^j)$ and $lst(\tau_i^j)$ first. Let S be the set of unoccupied time intervals that overlap the interval $[est(\tau_i^j), lst(\tau_i^j)]$ in the linked list. The unoccupied time intervals in S are collected by going through the list. Each time when a pair of time slots $(w, w+1)$ is examined, we compute $\ell = \max\{est(\tau_i^j), ft(w)\}$ and $\mu = \min\{lst(\tau_i^j), st(w+1)\}$, where $ft(w)$ is the finish time of the time slot w, and $st(w+1)$ is the start time of the slot next to w. If $\ell \leq \mu$, then we add the interval $[\ell, \mu]$ to S.

The free intervals in S are the potential time slots which τ_i^j can be assigned to. Since we try to schedule τ_i^j as close to one period away from the preceding instance τ_i^{j-1} as possible, we sort S, based on the function of the lower bound of each interval, $\alpha(s_i^{j-1} + p_i - \ell)$, in ascending order. Without loss of generality, we assume that S after the sorting is denoted by $\{int_1, int_2, \ldots, int_{|S|}\}$

The idea is that if τ_i^j is scheduled to int_k, then the value in equation 21 will be smaller than that of the case in which τ_i^j is scheduled to int_{k+1}.

The scheduling of τ_i^j can be described as follows. Starting from int_1, we check whether the length of the interval is greater or equal to the execution time of τ_i^j or not. If yes, then we schedule the instance to the interval. One new time slot is created in which the start time is the lower bound of the interval and the finish time equals the start time plus the execution time. The created time slot is added to the linked list and the scheduling is done. If the length is smaller than the execution time, then we check the length of the next interval until all intervals are examined. An example is shown in Figure 3 in which the slot with dark area represents τ_i^j. In this example we assume that $est(\tau_i^j) \leq f_1$ and $s_2 - f_1 > e$. It means the free slot between the first and second occupied slots can be assigned to τ_i^j.

4.2 Sliding of the Time Slots

In case none of the intervals in S can accommodate a task instance, the sliding technique is used to create a big enough interval by sliding the existence time slots in the list.

To make the sliding technique work, we maintain two values for each time slot: *left laxity* and *right laxity*. The value of left laxity indicates the amount of time units by which a time slot can be left-shifted to a earlier start time. Similarly, the right laxity indicates the amount of time units by which a time slot can be right-shifted to a later start time.

Given the time slots w_{k-1}, w_k, and w_{k+1}, where a and b are the task and instance identifiers of w_k respectively, the laxity values of the time slot w_k can be computed by:

$$\begin{aligned} left\_laxity(w_k) &= min\{s_a^b - est',\ s_a^b - ft(w_{k-1}) + left\_laxity(w_{k-1})\} && (22) \\ right\_laxity(w_k) &= min\{lst' - s_a^b,\ st(w_{k+1}) - f_a^b + right\_laxity(w_{k+1})\} && (23) \\ \text{where} \quad & est' = max\{est(\tau_a^b),\ s_a^{b+1} - (p_a + \eta_a)\} \\ \text{and} \quad & lst' = min\{lst(\tau_a^b),\ s_a^{b+1} - (p_a - \lambda_a)\}. \end{aligned}$$

Note that the interval $[est', lst']$ defines the sliding range during which τ_a^b can start without shifting τ_a^{b-1} or τ_a^{b+1}. A schematic illustration of equations 22 and 23 is given in Figure 4.

From equations 22 and 23, we see that the computing of *left_laxity*(w_k) depends on that of w_{k-1} and the computing of $right\_laxity(w_k)$ depends on that of w_{k+1}. It implies a two-pass computation

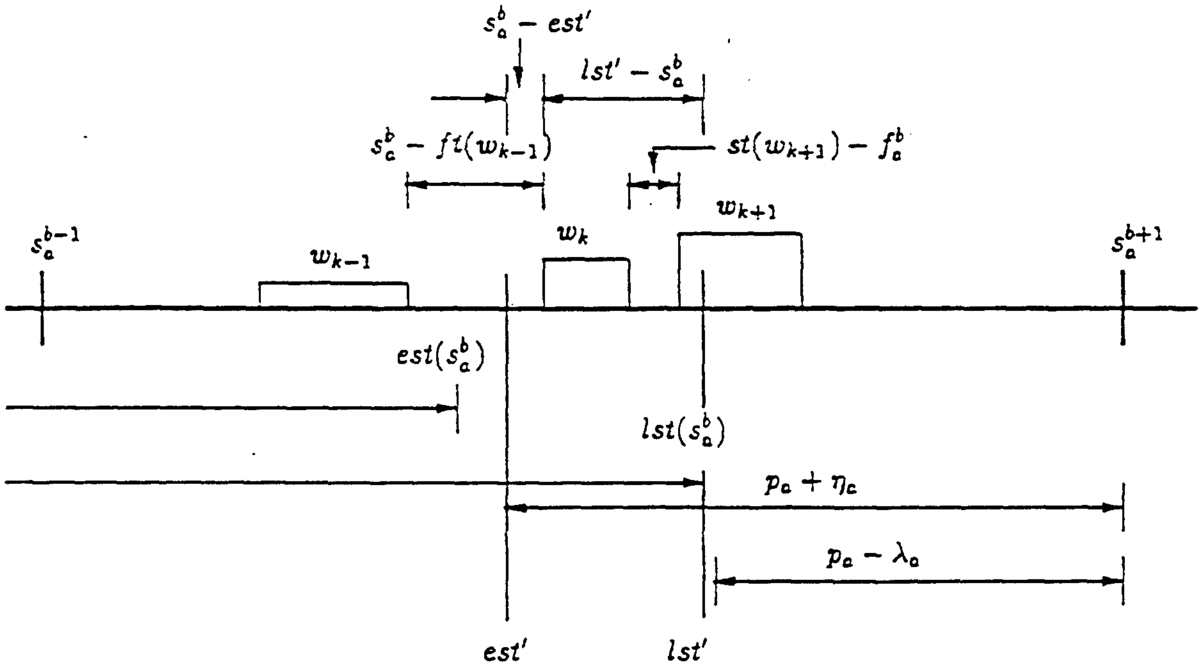


Figure 4: An illustration of $left\_laxity(w_k)$ and $right\_laxity(w_k)$

is needed to compute the laxity values for all time slots. The complexity is $O(2N)$ where N is the number of time slots in the linked list.

The basic idea of the sliding technique is described as follows. Given a task instance τ_i^j and a set of unoccupied intervals, $S = \{int_1, int_2, \ldots, int_{|S|}\}$, we check one interval at a time to see if the interval can be enlarged by shifting the existent time slots. Two possible ways of enlargement are (1) by either shifting the time slots, that precede the interval, to the left or (2) shifting the slots, that follow the interval, to the right. The shifting depends on which direction minimizes the objective function in Equation 21.

4.3 The Algorithm

An algorithmic description about how to schedule a task instance, as described in Sections 4.1 and 4.2, is given in Table 3.

The procedures Left_Shift(w_k,time_units) and Right_Shift(w_k,time_units) in Table 3 may involve the shifting of more than one time slot recursively. For example, consider the case in Figure 4, if Right_Shift(w_k,$lst' - s_a^b$) is invoked (i.e. w_k is to be shifted right by $lst' - s_a^b$ time units), then w_{k+1} has to be shifted too. It is because the gap between w_k and w_{k+1} is $st(w_{k+1}) - f_a^b$ which is smaller than $lst' - s_a^b$. In this case, Right_Shift(w_{k+1},$lst' - s_a^b - st(w_{k+1}) + f_a^b$) is invoked.

We do not enlarge an interval at both ends. Enlarging an interval at both ends needs to shift certain amount of preceding time slots to the left and shift some succeeding slots to the right. It is possible that some task instance τ_x^y is shifted left, while τ_x^{y+1} is shifted right. As a consequence, the timing constraints between s_x^y and s_x^{y+1} could be violated. For example, Let s_x^y and s_x^{y+1} before the shifting be 10 and 20 respectively. The execution time for τ_x is 5 time units. Assume the *left laxity* of τ_x^y is 5 and the *right laxity* of τ_x^{y+1} is 5. It implies $s_x^{y+1} - s_x^y \leq 15$. Consider the scheduling of a task instance τ_i^j with execution time 15. If we enlarge the interval between τ_x^y and τ_x^{y+1} by shifting τ_x^y left 5 time units and τ_x^{y+1} right 5 time units, then we get a new interval with 15 time units for τ_i^j. However, it turns out that $s_x^{y+1} = 25$, $s_x^y = 5$, and the relative timing constraints between τ_x^y and τ_x^{y+1} is violated.

5 The Priority-Based Scheduling of a Task Set

We consider the priority-based algorithms for scheduling a set of periodic tasks with hybrid timing constraints. Given a set of periodic tasks $\Gamma = \{ \tau_i \mid i = 1, \ldots, n \}$ with the task characteristics described in Section 2, we compute the LCM of all periods. Each task τ_i is extended to n_i task instances: $\tau_i^1, \tau_i^2, \ldots, \tau_i^{n_i}$. A scheduling algorithm σ for Γ is to totally order the instances of all tasks within the LCM. Namely, σ : *task_id* × *instance_id* → *integer*.

Three algorithms are considered. They are *smallest latest-start-time first* (SLsF), *smallest period first* (SPF), and *smallest jitter first* (SJF) algorithms.

5.1 SLsF

The scheduling window for a task instance τ_i^j depends on the scheduling of its preceding instance. Once s_i^{j-1} is determined, the scheduling window of the instance can be computed by equations 5 and 6. The scheduling window for the first instance of a task τ_i is defined as $[r_i, d_i - e_i]$.

The idea of the SLsF algorithm is to pick one candidate instance with the minimum *lst* among all tasks at a time. One counter for each task is maintained to indicate the candidate instance. All counters are initialized to 1. Each time when a task instance with the smallest *lst* is chosen, the algorithm in Table 3 is invoked to schedule the instance. After the scheduling of the instance is done, the counter is increased by one. The counter for τ_i overflows when it reaches $n_i + 1$. It means that all the instances of τ_i are scheduled. The algorithm terminates when all counters overflow.

We can compute the relative deadline for a task instance by adding the execution time to the *lst*. If the execution times for all tasks are identical, the SLsF algorithm is equivalent to the *earliest deadline first* (EDF) algorithm.

5.2 SPF

The task periods determine the LCM of Γ and the numbers of instances for tasks within the LCM. In the most cases, the task with the smaller period has the tighter timing constraints. Namely, $(\lambda_i + \eta_i) \leq (\lambda_j + \eta_j)$ if $p_i \leq p_j$. To make the tasks with the smaller periods meet their timing constraints, the SPF algorithm favors the tasks with smaller periods.

The SPF algorithm uses the period as the key to arrange all tasks in non-decreasing order. The task with the smallest period is selected to schedule first. The instances of a particular task are scheduled one by one by invoking the algorithm in Table 3. After all the instances of a task are scheduled, the next task in the sequence is scheduled.

5.3 SJF

We define the *jitter* of a task τ_i as $(\lambda_i + \eta_i)$. It is proportional to the range of the scheduling window. Hence, The schedulability of a task also depends on the jitter.

Instead of using the period as the measurement, the SJF algorithm assigns the higher priority to the tasks with the smaller jitters. The task with the smallest jitter is scheduled first.

5.4 The Solution

The composition of the time-based scheduling of a task instance and the priority assignment of task instances is shown in Figure 5. The priority assignment can be done by using SLsF, SPF, or SJF. The function *Schedule_An_Instance()* is invoked to schedule a single task instance.

6 Experimental Evaluation

We conduct two experiments to study and compare the performance of the three algorithms. The purpose of the first experiment is to study the effect of the number of tasks and utilization on

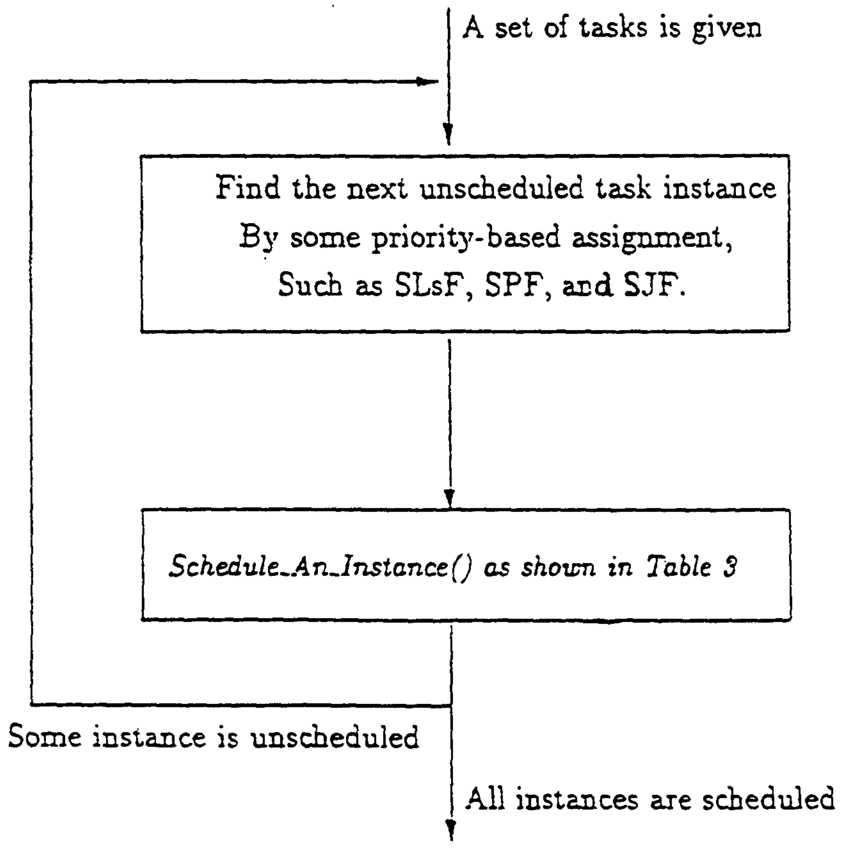

A set of tasks is given
Find the next unscheduled task instance
By some priority-based assignment,
Such as SLsF, SPF, and SJF.
Schedule_An_Instance() as shown in Table 3
Some instance is unscheduled
All instances are scheduled

Figure 5: A schematic flowchart for the solution

the schedulability of each algorithm. The objective of the second experiment is to compare the performance of the three algorithms.

6.1 The First Experiment

The task generation scheme for the first experiment is characterized by the following parameters.

- Periods of the tasks: We consider a homogeneous system in which the period of one task could be either the same as or multiple of the period of another. We consider a system with 40, 80, 160, 320, and 640 as the candidate periods. There may be more than one task with the same period.
- The execution time of a task, e_i : It has the uniform distribution over the range $[0,\frac{p_i}{16}]$, where p_i is the period of the task τ_i. The execution time could be a real value.
- The jitters of a task: $\lambda_i = \eta_i = 0.1 \times p_i$.

We define the utilization of a task system as

$$\sum_{i=1}^{N} \frac{e_i}{p_i} \tag{24}$$

In the first experiment, the utilization value and the number of tasks in a set are the controlled variables. Given an utilization value U and the number of tasks N the scheme first generates a run of raw data by randomly generating a set of N tasks based on the the selected periods, jitter values, and the execution time distribution. The utilization of the raw data, u, is then computed by Equation 24. Finally, the utilization value of the raw data is scaled up or down to U by multiplying $\frac{U}{u}$ to the execution time of each generated task. As a consequence, we obtain a set of tasks with the specified (U,N) value.

For each combination of (U,N) in which U = 5%, 10%, 15%, ... 100% and N = 10, 20, and 30, we apply the scheme to generate 5000 cases of input data and use the three algorithms to solve them. The schedulability degree of each (U,N) combination for an algorithm is obtained by dividing the number of solved cases by 5000. Since the jitter values is 1/10 of periods, it is observed that the SPF and SJF algorithms yield the same results. The results are shown in Figure 6.

As can be seen in Figures 6(a) and (b) the number of tasks has the different effects on the three algorithms. For SLsF, given a fixed utilization value, the schedulability degree increases

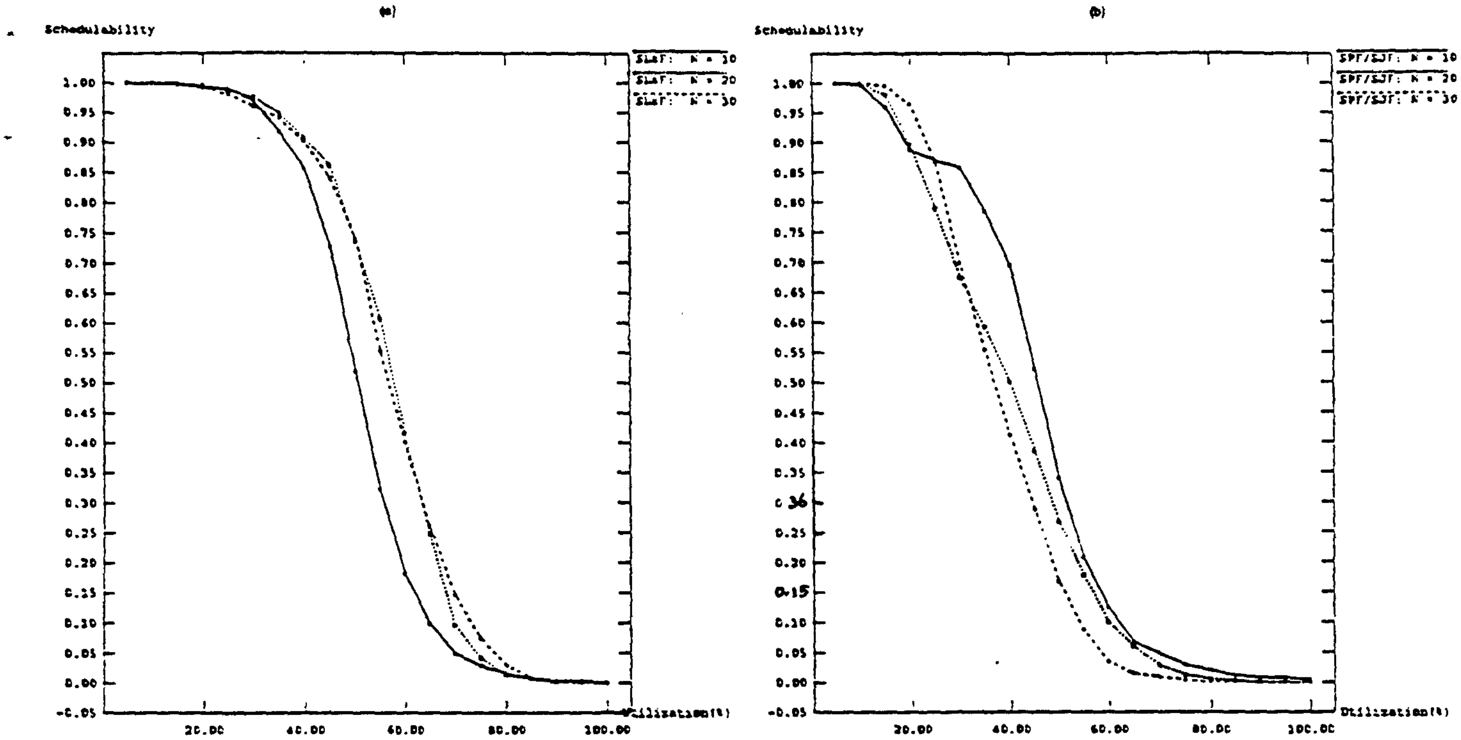


Figure 6: The effect of the numbers of tasks on the schedulability

as the number of tasks in a system becomes bigger. It is beacuse the execution time of a task becomes smaller as the number of tasks increases. For a task system with smaller execution time distribution, the chance for SLsF to find a feasible solution is bigger. The same phenomenon is also found in Figure 6(b) for SPF and SJF in the low-utilization cases (i.e. $U \leq 20\%$). However, for the high-utilization cases in Figure 6(b), the complexity of the number of tasks dominates the algorithms and the schedulability decreases.

6.2 The Second Experiment

The task generation scheme for the second experiment is characterized by the following parameters.

- LCM = 300
- The number of tasks is 20.
- Periods of the tasks: We consider the factors of the LCM as the periods. They are 20, 30, 50, 60, 100, 150, and 300. There may be more than one task with the same period.

- The execution time of a task, e_i : It has the uniform distribution over the range $[0, \frac{p_i}{15}]$, where p_i is the period of the task τ_i. The execution time could be a real value.

- The jitters of a task: $\lambda_i = \eta_i = 0.1 \times p_i + 2 \times e_i$.

The generation scheme for the second experiment is similar to the first one. Given an utilization value U, a set of 20 tasks is randomly generated according to the parameters listed above and then the execution time of each task is normalized in order to make the utilization value equal to U exactly.

We generate 5000 cases of different task sets for each utilization value ranging from 0.05 to 1.00. The schedulability degree of each algorithm on a particular utilization value is obtained by dividing the number of solved cases by 5000. We compare the schedulability degrees of the algorithms on different utilization values. The results are shown in Figure 7(a).

As can be see in Figure 7(a) the SLsF algorithm outperforms the other two algorithms. For example, when the utilization = 50%, the schedulability degree of SLsF is 0.575 while those of SPF and SJF are less than 0.2. It is because the way of assigning the priorities to the task instances in the SLsF algorithm reflects the urgency of task instances by considering the latest start times.

We also compare the objective function value π in Equation 21 among the three algorithms. We define the normalized objective function for an algorithm as

$$\hat{z} = \sum_{i=1}^{5000} \frac{z_i}{5000}, \qquad (25)$$

$$\text{where } \; z_i = \begin{cases} 1 & \text{if the algorithm can not find a feasible solution to case } i. \\ 0 & \text{if } max(i) = min(i). \\ \frac{\pi_i - min(i)}{max(i) - min(i)} & \text{otherwise.} \end{cases}$$

Given case i, the values of $min(i)$ and $max(i)$ are calculated among the objective values obtained from the algorithms which solve the case. For the algorithms which can not find a feasible solution to case i, the objective values are not taken into account when $min(i)$ and $max(i)$ are calculated. The results of the normalized objective functions for each algorithm on different utilization values are shown in Figure 7(b).

It is observed that in the low-utilization cases SJF finds feasible solutions with smaller objective values. It is because that SJF schedules the tasks with the smallest jitters first. By scheduling the tasks with smaller jitter value first it is more easier to make the instances of a task one period apart. we can find a feasible solution with smaller objective value. However, in the middle- or

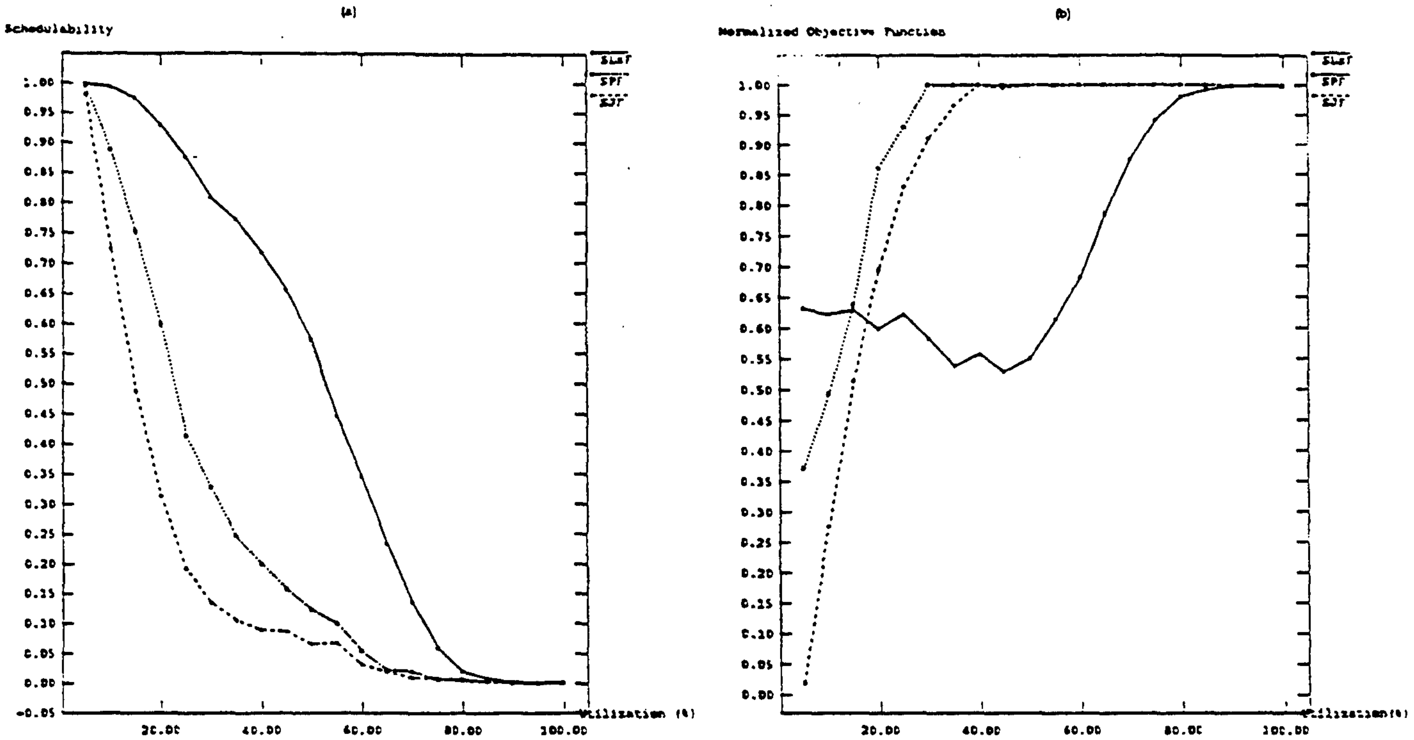


Figure 7: The comparison of three algorithms

high-utilization cases, the schedulability dominates the normalized objective function, and SLsF outperforms the other two algorithms in these regions.

7 Summary

In this paper we have considered the static non-preemptive scheduling algorithm on a single processor for a set of periodic tasks with hybrid timing constraints. The time-based scheduling algorithm is used to schedule a task instance once the scheduling window of the instance is given. We also have presented three priority assignment algorithms for the task instances and conducted experiments to compare the performance. From the experimental results, we see that the SLsF outperforms the other two algorithms.

The techniques presented in this chapter can be applied to multi-processor real-time systems. Communication and synchronization constraints can be also incorporated. In our future work, the extension to a distributed computing systems will be investigated.

References

[1] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. Arinc 659 scheduling: Problem definition. In *Proceedings of IEEE Real-Time Systems Symposium*, San Juan, PR, Dec. 1994.

[2] M. L. Dertouzos and A.K. Mok. Multiprocessor on-line scheduling of hard real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, Dec. 1989.

[3] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 116–128, Dec. 1991.

[4] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 108–115, Paris, France, 1990.

[5] T. Shepard and J.A.M. Gagne. A pre-run-time scheduling algorithm for hard real-time systems. *IEEE Transactions on Software Engineering*, 17(7):669–677, July 1991.

[6] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2), March 1994.

[7] J.P.C. Verhoosel, E.J. Luit, D.K. Hammer, and E. Jansen. A static scheduling algorithm for distributed hard real-time systems. *Real-Time Systems*, pages 227–246, 1991.

[8] J. Xu and D.L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.

```
Schedule_An_Instance ($\tau_i^j$):
Input: A linked list, a task instance $\tau_i^j$ and a sequence of sorted free intervals, $S = \{ int_1, int_2, \ldots, int_{|S|} \}$,
        in which each interval overlaps $[est(\tau_i^j), lst(\tau_i^j)]$.

Let the execution time of $\tau_i^j$ be $e$.
For $n = 1$ to $|S|$ do
    Let $int_n$ be $[\ell,\mu]$.
    If $\mu - \ell \geq e$ then
        Return a new time slot with start time $= \ell$ and finish time $= \ell + e$.
    End if.
End for.
Compute left laxity and right laxity for each time slot in the linked list by equations 22 and 23.
For $n = 1$ to $|S|$ do
    Let $int_n$ be $[\ell,\mu]$.
    If $\ell \geq s_i^{j-1} + p_i$ then /* Try left shift first then right shift */
        Let the time slot that immediately precedes $int_n$ be $w_k$.
        If $left\_laxity(w_k) + \mu - \ell \geq e$ then /* Left shift */
            Left_Shift($w_k, e - \mu + \ell$).
            Return a new time slot with start time $= \mu - e$ and finish time $= \mu$.
        Else
            Let the time slot that immediately follows $int_n$ be $w_k$.
            If $right\_laxity(w_k) + \mu - \ell \geq e$ then /* Right shift */
                Right_Shift($w_k, e - \mu + \ell$).
                Return a new time slot with start time $= \ell$ and finish time $= \ell + e$.
            End If.
        End If.
    Else /* Try right shift first then left shift */
        Let the time slot that immediately follows $int_n$ be $w_k$.
        If $right\_laxity(w_k) + \mu - \ell \geq e$ then /* Right shift */
            Right_Shift($w_k, e - \mu + \ell$).
            Return a new time slot with start time $= \ell$ and finish time $= \ell + e$.
        Else
            Let the time slot that immediately precedes $int_n$ be $w_k$.
            If $left\_laxity(w_k) + \mu - \ell \geq e$ then /* Left shift */
                Left_Shift($w_k, e - \mu + \ell$).
                Return a new time slot with start time $= \mu - e$ and finish time $= \mu$.
            End If.
        End If.
    End If.
End for.
Schedule $\tau_i^j$ at the end of linked list.
```

Table 3: The Scheduling of a Task Instance

REPORT DOCUMENTATION PAGE	Form approved OMB No 074-0188

1. AGENCY USE ONLY (leave blank)	2. REPORT DATE 1/3/1995	3. REPORT TYPE END DATES COVERED Technical Reports

4. TITLE AND SUBTITLE
Scheduling of Periodic Tasks with Relative Timing Constraints

5. FUNDING NUMNBERS
N00014-91-C-0195

6. AUTHOR(S)
S.-T. Cheng and A. K. Agrawala

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Maryland
Department of Computer Science
A.V. Williams Building
College Park, MD 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
CSTR 3392

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)
PhillipsLaboratory
Director of Contracting
3651 Lowry Avenue SE
Kirtland AFB, NM 87117-5777

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12.a. DISTRIBUTION/ AVAILABILITY STATEMENT

12.b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The problem of non-preemptive scheduling of a set of periodic tasks on a single processor has been traditionally considering the ready time and deadline on each task. As a consequence, a feasible schedule finds that in each period one instance of each task starts the execution after the ready time and completes the execution before the deadline. Recently, the timing requirements of the real-time systems emerge that the relative timing constraints are imposed on the consecutive executions of each task. In this paper, we consider the scheduling problem of the periodic tasks with the relative timing constraints imposed on two consecutive executions of a task. We analyze the timing constraints and derive the scheduling window for each task instance. Based on the scheduling window, we present the time-based approach of scheduling a task instance. The task instances are scheduled one by one based on their priorities assigned by the proposed algorithms in this paper. We conduct the experiments to compare the schedulability of the algorithms.

14. SUBJECT TERMS
Time constraints, timing requirements

15. NUMBER OF PAGES
21

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Umlimited

A Scalable Virtual Circuit Routing Scheme for ATM Networks*

Cengiz Alaettinoğlu
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292

Ibrahim Matta A. Udaya Shankar
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742

October 1994

Abstract

High-speed networks, such as ATM networks, are expected to support diverse quality-of-service (QoS) requirements, including real-time QoS. Real-time QoS is required by many applications such as voice and video. To support such service, routing protocols based on the Virtual Circuit (VC) model have been proposed. However, these protocols do not scale well to large networks in terms of storage and communication overhead.

In this paper, we present a scalable VC routing protocol. It is based on the recently proposed viewserver hierarchy, where each viewserver maintains a partial view of the network. By querying these viewservers, a source can obtain a merged view that contains a path to the destination. The source then sends a request packet over this path to setup a real-time VC through resource reservations. The request is blocked if the setup fails. We compare our protocol to a simple approach using simulation. Under this simple approach, a source maintains a full view of the network. In addition to the savings in storage, our results indicate that our protocol performs close to or better than the simple approach in terms of VC carried load and blocking probability over a wide range of real-time workload.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*packet networks; store and forward networks*; C.2.2 [Computer-Communication Networks]: Network Protocols—*protocol architecture*; C.2.m [Routing Protocols]; F.2.m [Computer Network Routing Protocols].

* This work is supported in part by ARPA and Philips Labs under contract DASG60-92-0055 to Department of Computer Science, University of Maryland, and by National Science Foundation Grant No. NCR 89-04590. The work of C. Alaettinoğlu is also supported by National Science Foundation Grant No. NCR 93-21043. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, PL, the National Science Foundation, or the U.S. Government.

Contents

1 Introduction

Integrated services packet-switched networks, such as Asynchronous Transfer Mode (ATM) networks [21], are expected to carry a wide variety of applications with heterogeneous quality of service (QoS) requirements. For this purpose, new resource allocation algorithms and protocols have been proposed, including link scheduling, admission control, and routing. Link scheduling defines how the link bandwidth is allocated among the different services. Admission control defines the criteria the network uses to decide whether to accept or reject a new incoming application. Routing concerns the selection of routes to be taken by application packets (or cells) to reach their destination. In this paper, we are mainly concerned with routing for real-time applications (e.g., voice, video) requiring QoS guarantees (e.g., bandwidth and delay guarantees).

To provide real-time QoS support, a number of *virtual-circuit* (VC) routing approaches have been proposed. A *simple* (or straightforward) approach to VC routing is the link-state full-view approach. Here, each end-system maintains a view of the whole network, i.e. a graph with a vertex for every node[1] and an edge between two neighbor nodes. QoS information such as delay, bandwidth, and loss rate are attached to the vertices and the edges of the view. This QoS information is flooded regularly to all end-systems to update their views. When a new application requests service from the network, the source end-system uses its current view to select a *source route* to the destination end-system that is likely to support the application's requested QoS, i.e., a sequence of node ids starting from the source end-system and ending with the destination end-system. A VC-setup message is then sent over the selected source route to try to reserve the necessary resources (bandwidth, buffer space, service priority) and establish a VC.

Typically, at every node the VC-setup message visits, a set of admission control tests are performed to decide whether the new VC, if established, can be guaranteed its requested QoS without violating the QoS guaranteed to already established VCs. At any node, if these admission tests are passed, then resources are reserved and the VC-setup message is forwarded to the next node. On the other hand, if the admission tests fail, a VC-rejected message is sent back towards the source node releasing resource reservations made by the VC-setup message, and the application request is either blocked or another source route is selected and tried. If the final admission tests at the destination node are passed, then a VC-established message is sent back towards the source node confirming resource reservations made during the forward trip of the VC-setup message. Upon receiving the VC-established message, the application can start transmitting its packets over its

[1] We refer to switches and end-systems collectively as nodes.

reserved VC. This VC is torn down and resources are released at the end of the transmission.

Clearly, the above simple routing scheme does not scale up to large networks. The storage at each end-system and the communication cost are proportional to $N \times d$, where N is the number of nodes and d is the average number of neighbors to a node.

A traditional solution to this scaling problem is the area hierarchy used in routing protocols such as the Open Shortest Path First (OSPF) protocol [18]. The basic idea is to aggregate nodes hierarchically into areas: "close" nodes are aggregated into level 1 areas, "close" level 1 areas are aggregated into level 2 areas, and so on. An end-system maintains a view that contains the nodes in the same level 1 area, the level 1 areas in the same level 2 area, and so on. Thus an end-system maintains a smaller view than it would in the absence of hierarchy. Each area has its own QoS information derived from that of the subareas. A major problem of an area-based scheme is that aggregation results in loosing detailed link-level QoS information. This decreases the chance of the routing algorithm to choose "good" routes, i.e. routes that result in high successful VC setup rate (or equivalently high carried VC load).

Our scheme

In this paper, we present a scalable VC routing scheme that does not suffer from the problems of areas. Our scheme is based on the viewserver hierarchy we recently proposed in [3, 2] for large internetworks and evaluated for administrative policy constraints. Here, we are concerned with the support of performance/QoS requirements in large wide-area ATM-like networks, and we adapt our viewserver protocols accordingly.

In our scheme, views are not maintained by every end-system but by special switches called *viewservers*. For each viewserver, there is a subset of nodes around it, referred to as the viewserver's *precinct*. The viewserver only maintains the view of its precinct. This solves the scaling problem for storage requirement.

A viewserver can provide source routes for VCs between source and destination end-systems in its precinct. Obtaining a route between a source and a destination that are not in any single view involves accumulating the views of a sequence of viewservers. To make this process efficient, viewservers are organized hierarchically in levels, and an associated addressing structure is used. Each end-system has a set of addresses. Each *address* is a sequence of viewserver ids of decreasing levels, starting at the top level and going towards the end-system. The idea is that when the views of the viewservers in an address are merged, the merged view contains routes to the end-system from the top level viewservers.

We handle dynamic topology changes such as node/link failures and repairs, and link cost changes. Nodes detect topology changes affecting itself and neighbor nodes. Each node communicates these changes by flooding to the viewservers in a specified subset of nodes; this subset is referred to as its *flood area*. Hence, the number of packets used during flooding is proportional to the size of the flood area. This solves the scaling problem for the communication requirement.

Thus our VC routing protocol consists of two subprotocols: a *view-query protocol* between end-systems and viewservers for obtaining merged views; and a *view-update protocol* between nodes and viewservers for updating views.

Evaluation

In this paper, we compare our viewserver-based VC routing scheme to the simple scheme using VC-level simulation. In our simulation model, we define network topologies, QoS requirements, viewserver hierarchies, and evaluation measures. To the best of our knowledge, this is the first evaluation of a dynamic hierarchical-based VC routing scheme under real-time workload.

Our evaluation measures are the amount of memory required at the end-systems, the amount of time needed to construct a path[2], the carried VC load, and the VC blocking probability. We use network topologies each of size 2764 nodes. Our results indicate that our viewserver-based VC routing scheme performs close to or better than the simple scheme in terms of VC carried load and blocking probability over a wide range of workload. It also reduces the amount of memory requirement by up to two order of magnitude.

Organization of the paper

In Section 2, we survey recent approaches to VC routing. In Section 3, we present the view-query protocol for static network conditions, that is, assuming all links and nodes of the network remain operational. In Section 4, we present the view-update protocol to handle topology changes. In Section 5, we present our evaluation model. Our results are presented in Section 6. Section 7 concludes the paper.

[2] We use the terms route and path interchangeably.

2 Related Work

In this section, we discuss routing protocols recently proposed for packet-switched QoS networks. These routing protocols can be classified depending on whether they help the network support *qualitative* QoS or *quantitative* (real-time) QoS. For a qualitative QoS, the network tries to provide the service requested by the application with no performance guarantees. Such a service is often identified as "best-effort". A quantitative QoS provides performance guarantees (typically required by real-time applications); for example, an upper bound on the end-to-end delay for any packet received at the destination.

Routing protocols that make routing decisions on a per VC basis can be used to provide either qualitative or quantitative QoS. For a quantitative QoS, some admission control tests should be performed during the VC-setup message's trip to the destination to try to reserve resources along the VC's path as described in Section 1.

On the other hand, the use of routing protocols that make routing decisions on a per packet basis is problematic in providing resource guarantees [5], and qualitative QoS is the best service the network can offer.

Since we are concerned in this paper with real-time QoS, we limit our following discussion to VC routing schemes proposed or evaluated in this context. We refer the reader to [19, 6] for a good survey on many other routing schemes.

Most of the VC routing schemes proposed for real-time QoS networks are based on the link-state full-view approach described in Section 1 [6, 1, 10, 24]. Recall that in this approach, each end-system maintains a view of the whole network, i.e. a graph with a vertex for every node and an edge between two neighbor nodes. QoS information is attached to the vertices and the edges of the view. This QoS information is distributed regularly to all end-systems to update their views and thus enable the selection of appropriate source routes for VCs, i.e. routes that are likely to meet the requested QoS. The proposed schemes mainly differ in how this QoS information is used. Generally, a cost function is defined in terms of the QoS information, and used to estimate the cost of a path to the VC's destination. The route selection algorithm then favors short paths with minimum cost. See [17, 22] for an evaluation of several schemes.

A number of VC routing schemes have also been designed for networks using the Virtual Path (VP) concept [15, 14]. This VP concept has been proposed to simplify network management and control by having separate (logically) fully-connected subnetworks, typically one for each service class. In each VP subnetwork, simple routing schemes that only consider one-hop and two-hop paths are used. However, the advantage of using VPs can be offset by a decrease in statistical multiplexing gains of the subnetworks [15]. In this work, we are interested in *general* network topologies, where the shortest paths can be of arbitrary hop length and the overhead of routing protocols is of much concern.

All the above VC routing schemes are based on the link-state approach. VC routing schemes based on the path-vector approach have also been proposed [13]. In this approach, for each destination a node maintains a set of paths, one through each of its neighbor nodes. QoS information is attached to these paths. For each destination, a node exchanges its best feasible path[3] with its neighbor nodes. The scheme in [13] provides two kinds of routes: pre-computed and on-demand. Pre-computed routes match some well-known QoS requirements, and are maintained using the path-vector approach. On-demand routes are calculated for specific QoS requirements upon request. In this calculation, the source broadcasts a special packet over all candidate paths. The destination then selects a feasible path from them and informs the source [13, 23]. One drawback of this scheme is that obtaining on-demand routes is very expensive since there are potentially exponential number of candidate paths between the source and the destination.

The link-state approach is often proposed and favored over the path-vector approach in QoS architectures for several reasons [16]. An obvious reason is simplicity and complete control of the source over QoS route selection.

The above VC routing schemes do not scale well to large QoS networks in terms of storage and communication requirements. Several techniques to achieve scaling exist. The most common technique is the area hierarchy described in Section 1.

The landmark hierarchy [26, 25] is another approach for solving the scaling problem. The link-state approach can not be used with the landmark hierarchy. A thorough study of enforcing QoS and policy constraints with this hierarchy has not been done.

Finally, we should point out that extensive effort is currently underway to fully specify and standardize VC routing schemes for the future integrated services Internet and ATM networks [9].

3 Viewserver Hierarchy Query Protocol

In this section, we present our scheme for static network conditions, that is, all links and nodes remain operational. The dynamic case is presented in Section 4.

[3] A feasible path is a path that satisfies the QoS constraints of the nodes in the path.

Conventions: Each node has a unique id. NodeIds denotes the set of node-ids. For a node u, we use $nodeid(u)$ to denote the id of u. $NodeNeighbors(u)$ denotes the set of ids of the neighbors of u.

In our protocol, a node u uses two kinds of sends. The first kind has the form "Send(m) to v", where m is the message being sent and v is the destination-id. Here, nodes u and v are neighbors, and the message is sent over the physical link (u, v). If the link is down, we assume that the packet is dropped.

The second kind of send has the form "Send(m) to v using sr", where m and v are as above and sr is a source route between u and v. We assume that as long as there is a sequence of up links connecting the nodes in sr, the message is delivered to v. This requires a transport protocol support such as TCP [20].

To implement both kind of sends, we assume there is a reserved VC on each link for sending routing, signaling and control messages [4]. This also ensures that routing messages do not degrade the QoS seen by applications.

Views and Viewservers

Views are maintained by special nodes called *viewservers*. Each viewserver has a *precinct*, which is a set of nodes around the viewserver. A viewserver maintains a *view*, consisting of the nodes in its precinct, links between these nodes and links outgoing from the precinct[4]. Formally, a viewserver x maintains the following:

$Precinct_x$ $\subseteq$ NodeIds. Nodes whose view is maintained.

$View_x$. View of x.

$$= \{\langle u,\ timestamp,\ expirytime,\ \{\langle v,\ cost\rangle : v \in NodeNeighbors(u)\}\rangle :$$
$$u \in Precinct_x\}$$

The intention of $View_x$ is to obtain source routes between nodes in $Precinct_x$. Hence, the choice of nodes to include in $Precinct_x$ and the choice of links to include in $View_x$ are not arbitrary. $Precinct_x$ and $View_x$ must be connected; that is, between any two nodes in $Precinct_x$, there should be a path in $View_x$. Note that $View_x$ can contain links to nodes outside $Precinct_x$. We say that a node u is *in the view* of a viewserver x, if either u is in the precinct of x, or $View_x$ has a link from a node in the precinct of x to node u. Note that the precincts and views of different viewservers can be overlapping, identical or disjoint.

[4] Not all the links need to be included.

For a link (u, v) in the view of a viewserver x, $View_x$ stores a *cost*. The cost of the link (u, v) equals a vector of values if the link is known to be up; each cost value estimates how expensive it is to cross the link according to some QoS criteria such as delay, throughput, loss rate, etc. The cost equals ∞ if the link is known to be down. Cost of a link changes with time (see Section 4). The view also includes *timestamp* and *expirytime* fields which are described in Section 4.

Viewserver Hierarchy

For scaling reasons, we cannot have one large view. Thus, obtaining a source route between a source and a destination which are far away, involves accumulating views of a sequence of viewservers. To keep this process efficient, we organize viewservers hierarchically. More precisely, each viewserver is assigned a hierarchy level from $0, 1, \ldots$, with 0 being the top level in the hierarchy. A parent-child relationship between viewservers is defined as follows:

1. Every level i viewserver, $i > 0$, has a parent viewserver whose level is less than i.
2. If viewserver x is a parent of viewserver y then x's precinct contains y and y's precinct contains x.
3. The precinct of a top level viewserver contains all other top level viewservers.

In the hierarchy, a parent can have many children and a child can have many parents. We extend the range of the parent-child relationship to ordinary nodes; that is, if $Precinct_x$ contains the node u, we say that u is a child of x, and x is a parent of u. We assume that there is at least one parent viewserver for each node.

For a node u, an address is defined to be a sequence $\langle x_0, x_1, \ldots, x_t \rangle$ such that x_i for $i < t$ is a viewserver-id, x_0 is a top level viewserver-id, x_t is the id of u, and x_i is a parent of x_{i+1}. A node may have many addresses since the parent-child relationship is many-to-many. If a source node wants to establish a VC to a destination node, it first queries the name servers to obtain a set of addresses for the destination[5]. Second, it queries viewservers to obtain an accumulated view containing both itself and the destination node (it can reach its parent viewservers by using fixed source routes to them). Then, it chooses a feasible source route from this accumulated view and initiates the VC setup protocol on this path.

View-Query Protocol: Obtaining Source Routes

We now describe how a source route is obtained.

[5] Querying the name servers can be done in the same way as is done currently in the Internet.

We want a sequence of viewservers whose merged views contains both the source and the destination nodes. Addresses provide a way to obtain such a sequence, by first going up in the viewserver hierarchy starting from the source node and then going down in the viewserver hierarchy towards the destination node. More precisely, let $\langle s_0, \ldots, s_t \rangle$ be an address of the source. and $\langle d_0, \ldots, d_l \rangle$ be an address of the destination. Then, the sequence $\langle s_{t-1}, \ldots, s_0, d_0, \ldots, d_{l-1} \rangle$ meets our requirements. In fact, going up all the way in the hierarchy to top level viewservers may not be necessary. We can stop going up at a viewserver s_i if there is a viewserver $d_j, j < l$, in the view of s_i (one special case is where $s_i = d_j$).

The view-query protocol uses two message types:

- (RequestView, *s_address*, *d_address*)
  where *s_address* and *d_address* are the addresses for the source and the destination respectively. A RequestView message is sent by a source node to obtain an accumulated view containing both the source and the destination nodes. When a viewserver receives a RequestView message, it either sends back its view or forwards this request to another viewserver.
- (ReplyView, *s_address*, *d_address*, *accumview*)
  where *s_address* and *d_address* are as above and *accumview* is the accumulated view. A ReplyView message is sent by a viewserver to the source or to another viewserver closer to the source. The *accumview* field in a ReplyView message equals the union of the views of the viewservers the message has visited.

We now describe the view-query protocol in more detail (please refer to Figures 1 and 2). To establish a VC to a destination node, the source node sends a RequestView packet containing the source and the destination addresses to its parent in the source address.

Upon receiving a RequestView packet, a viewserver x checks if the destination node is in its precinct[6]. If it is, x sends back its view in a ReplyView packet. If it is not, x forwards the request packet to another viewserver as follows (details in Figure 2): x checks whether any viewserver in the destination address is in its view. If there is such a viewserver, x sends the RequestView packet to the last such one in the destination address. Otherwise x is a viewserver in the source address, and it sends the packet to its parent in the source address.

When a viewserver x receives a ReplyView packet, it merges its view to the accumulated view in the packet. Then it sends the ReplyView packet towards the source node in the same way it would send a RequestView packet towards the destination node (i.e. the roles of the source address

[6] Even though the destination can be in the view of x, its QoS characteristics is not in the view if it is not in the precinct of x.

```
Constants
  $FixedRoutes_u(x)$, for every viewserver-id $x$ such that $x$ is a parent of $u$,
      $= \{\langle y_1, \ldots, y_n \rangle : y_i \in$ NodeIds$\}$. Set of routes to $x$
Events
  $RequestView_u$(s_address, d_address)        {Executed when $u$ wants a source route}
    Let s_address be $\langle s_0, \ldots, s_{t-1}, s_t \rangle$, and $sr \in FixedRoutes_u(s_{t-1})$;
    Send(RequestView, s_address, d_address) to $s_{t-1}$ using $sr$

  $Receive_u$(ReplyView, s_address, d_address, accumview)
    Choose a feasible source route using accumview;
    If a feasible route is not found
        Execute $RequestView_u$ again with another source address and/or destination address
```

Figure 1: View-query protocol: Events and state of a source node u.

```
Constants
  $Precinct_x$. Precinct of $x$.
Variables
  $View_x$. View of $x$.
Events
  $Receive_x$(RequestView, s_address, d_address)
    Let d_address be $\langle d_0, \ldots, d_t \rangle$;
    if $d_t \notin Precinct_x$ then
    $forward_x$(RequestView, s_address, d_address, {});
    else $forward_x$(ReplyView, d_address, s_address, $View_x$);        {addresses are switched}
    endif

  $Receive_x$(ReplyView, s_address, d_address, view)
    $forward_x$(ReplyView, s_address, d_address, view $\cup$ $View_x$)

  where procedure $forward_x$(type, s_address, d_address, view)
    Let s_address be $\langle s_0, \ldots, s_t \rangle$, d_address be $\langle d_0, \ldots, d_l \rangle$;
    if $\exists i : d_i$ in $View_x$ then
    Let $i = \max\{j : d_j$ in $View_x\}$;
    $target := d_i$;
    else $target := s_i$ such that $s_{i+1} = nodeid(x)$;
    endif;
    $sr$ := choose a route  to $target$ from $nodeid(x)$ using $View_x$;
    if $type$ = RequestView then
    Send(RequestView, s_address, d_address) to $target$ using $sr$;
    else Send(ReplyView, s_address, d_address, view) to $target$ using $sr$;
    endif
```

Figure 2: View-query protocol: Events and state of a viewserver x.

and the destination address are interchanged).

When the source receives a ReplyView packet, it chooses a feasible path using the *accumview* in the packet. If it does not find a feasible path, it can try again using a different source and/or destination addresses. Note that the source does not have to throw away the previous accumulated views; it can merge them all into a richer accumulated view. In fact, it is easy to change the protocol so that the source can also obtain views of individual viewservers to make the accumulated view even richer. Once a feasible source route is found, the source node initiates the VC setup protocol.

Above we have described one possible way of obtaining the accumulated views. There are various other possibilities, for example: (1) restricting the ReplyView packet to take the reverse of the path that the RequestView packet took; (2) having ReplyView packets go all the way up in the viewserver-hierarchy for a richer accumulated view; (3) having the source poll the viewservers directly instead of the viewservers forwarding request/reply messages to each other; (4) not including non-transit nodes (e.g. end-systems) other than the source and the destination nodes in the *accumview*; (5) including some QoS requirements in the RequestView packet, and having the viewservers filter out some nodes and links.

4 Update Protocol for Dynamic Network Conditions

In this section, we first describe how topology changes such as link/node failures, repairs and cost changes, are detected and communicated to viewservers, i.e. the view-update protocol. Then, we modify the view-query protocol appropriately.

View-Update Protocol: Updating Views

Viewservers do not communicate with each other to maintain their views. Nodes detect and communicate topology changes to viewservers. Updates are done periodically and also optionally after a change in the outgoing link costs.

The communication between a node and viewservers is done by flooding over a set of nodes. This set is referred to as the *flood area*. The topology of a flood area must be a connected graph. For efficiency, the flood area can be implemented by a hop-count.

Due to the nature of flooding, a viewserver can receive information out of order from a node. In order to avoid old information replacing new information, each node includes successively increasing time stamps in the messages it sends. The *timestamp* field in the view of a viewserver equals the largest timestamp received from each node.

Due to node and link failures, communication between a node and a viewserver can fail, resulting in the viewserver having out-of-date information. To eliminate such information, a viewserver deletes any information about a node if it is older than a *time-to-die* period. The *expirytime* field in the view of a viewserver equals the end of the time-to-die period for a node. We assume that nodes send messages more often than the time-to-die value (to avoid false removal).

The view-update protocol uses one type of message as follows:

- (Update, *nid*, *timestamp*, *floodarea*, *ncostset*)
  is sent by the node to inform the viewservers about current costs of its outgoing links. Here, *nid* and *timestamp* indicate the id and the time stamp of the node, *ncostset* contains a cost for each outgoing link of the node, and *floodarea* is the set of nodes that this message is to be sent over.

Constants:
  $FloodArea_g$. ($\subseteq$ NodeIds). The flood area of the node.
Variables:
  $Clock_g$: Integer. Clock of g.

Figure 3: State of a node g.

The state maintained by a node g is listed in Figure 3. We assume that consecutive reads of $Clock_g$ returns increasing values.

Constants:
  $Precinct_x$. Precinct of x.
  $TimeToDie_x$: Integer. Time-to-die value.
Variables:
  $View_x$. View of x.
  $Clock_x$: Integer. Clock of x.

Figure 4: State of a viewserver x.

The state maintained by a viewserver x is listed in Figure 4.

The events of node g are specified in Figure 5. The events of a viewserver x are specified in Figure 6. When a viewserver x recovers, $View_x$ is set to $\{\}$. Its view becomes up-to-date as it receives new information from nodes (and remove false information with the time-to-die period).

```
$Update_g$        {Executed periodically and also optionally upon a change in outgoing link costs}
    $ncostset$ := compute costs for each outgoing link;
    $flood_g$((Update, $nodeid(g)$, $Clock_g$, $FloodArea_g$, $ncostset$));

$Receive_g(packet)$        {an Update packet}
    $flood_g(packet)$

where procedure $flood_g(packet)$
    if $nodeid(g) \in packet.floodarea$ then
    {remove $g$ from the flood area to avoid infinite exchange of the same message.}
    $packet.floodarea := packet.floodarea - \{nodeid(g)\}$;
    for all $h \in NodeNeighbors(g) \wedge h \in packet.floodarea$ do
    Send($packet$) to $h$;
    endif
```

Node Failure Model: A node can undergo failures and recoveries at anytime. We assume failures are fail-stop (i.e. a failed node does not send erroneous messages).

Figure 5: View-update protocol: Events of a node g.

```
$Receive_x$(Update, $nid$, $ts$, $FloodArea$, $ncset$)
    if $nid \in Precinct_x$ then
    if $\exists(nid, timestamp, expirytime, ncostset) \in View_x \wedge ts > timestamp$ then
    {received is more recent; delete the old one}
    delete $(nid, timestamp, expirytime, ncostset)$ from $View_x$;
    endif
    if $\neg\exists(nid, timestamp, expirytime, ncostset) \in View_x$ then
    $ncostset$ := subset of edge-cost pairs in $ncset$ that are in $View_x$;
    insert $(nid, ts, Clock_x + TimeToDie_x, ncostset)$ to $View_x$;
    endif
    endif
$Delete_x$        {Executed periodically to delete entries older than the time-to-die period}
    for all $(nid, tstamp, expirytime, ncset) \in View_x \wedge expirytime < Clock_x$ do
    delete $(nid, tstamp, expirytime, ncset)$ from $View_x$;
```

Viewserver Failure Model: A viewserver can undergo failures and recoveries at anytime. We assume failures are fail-stop. When a viewserver x recovers, $View_x$ is set to {}.

Figure 6: View update events of a viewserver x.

Changes to View-Query Protocol

We now enumerate the changes needed to adapt the view-query protocol to the dynamic case (the formal specification is omitted for space reasons).

Due to link and node failures, RequestView and ReplyView packets can get lost. Hence, the source may never receive a ReplyView packet after it initiates a request. Thus, the source should try again after a time-out period.

When a viewserver receives a RequestView message, it should reply with its views only if the destination node is in its precinct and its view contains a path to the destination. Similarly during forwarding of RequestView and ReplyView packets, a viewserver, when checking whether a node is in its view, should also check if its view contains a path to it.

5 Evaluation

In this section, we present the parameters of our simulation model. We use this model to compare our viewserver-based VC routing protocols to the simple approach. The results obtained are presented in Section 6.

Network Parameters

We model a campus network which consists of a campus backbone subnetwork and several department subnetworks. The backbone network consists of backbone switches and backbone links.

Each department network consists of a hub switch and several non-hub switches. Each non-hub switch has a link to the department's hub switch. And the department's hub switch has a link to one of the backbone switches. A non-hub switch can have links to other non-hub switches in the same department, to non-hub switches in other departments, or to backbone switches.

End-systems are connected to non-hub switches. An example network topology is shown in Figure 7.

In our topology, there are 8 backbone switches and 32 backbone links. There are 16 departments. There is one hub-switch in each department. There is a total of 240 non-hub switches randomly assigned to different departments. There are 2500 end-systems which are randomly connected to non-hub switches. Thus, we have a total of 2764 nodes.

In addition to the links connecting non-hub switches to the hub switches and hub switches to the backbone switches, there are 720 links from non-hub switches to non-hub switches in the same department, there are 128 links from non-hub switches to non-hub switches in different departments, and there are 64 links from non-hub switches to backbone switches.

The end-points of each link are chosen randomly. However, we make sure that the backbone network is connected; and there is a link from node u to node v iff there is a link from node v to

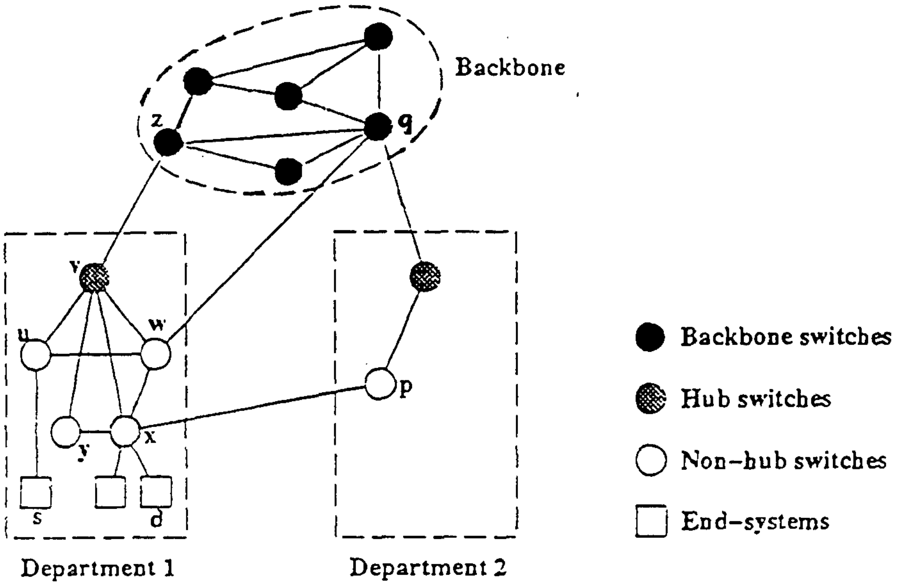


Figure 7: An example network topology.

node u.

Each link has a total of C units of bandwidth.

QoS and Workload Parameters

In our evaluation model, we assume that a VC requires the reservation of a certain amount of bandwidth that is enough to ensure an acceptable QoS for the application. This reservation amount can be thought of either as the peak transmission rate of the VC or its "effective bandwidth" [12] varying between the peak and average transmission rate.

VC setup requests arrive to the network according to a Poisson process of rate λ, each requiring one unit of bandwidth. Each VC, once it is successfully setup, has a lifetime of exponential duration with mean $1/\mu$. The source and the destination end-systems of a VC are chosen randomly.

An arriving VC is admitted to the network if at least one feasible path between its source and destination end-systems is found by the routing protocol, where a feasible path is one that has links with non-zero available capacity. From the set of feasible paths, a minimum hop path is used to establish the VC; one unit of bandwidth is allocated on each of its links for the lifetime of the VC. On the other hand, if a feasible path is not found, then the arriving VC is blocked and lost.

We assume that the available link capacities in the views of the viewservers are updated instantaneously whenever a VC is admitted to the network or terminates.

Viewserver Hierarchy Schemes

We have evaluated our viewserver protocol for several different viewserver hierarchies and query methods. We next describe the different viewserver schemes evaluated. Please refer to **Figure** 7 in the following discussion.

The first viewserver scheme is referred to as **base**. Each switch is a viewserver. A viewserver's precinct consist of itself and the neighboring nodes. The links in the viewserver's view consist of the links between the nodes in the precinct, and links outgoing from nodes in the precinct to nodes not in the precinct. For example, the precinct of viewserver u consists of nodes u, v, w, s.

As for the viewserver hierarchy, a backbone switch is a level 0 viewserver, a hub switch is a level 1 viewserver and a non-hub switch is a level 2 viewserver. Parent of a hub switch viewserver is the backbone switch viewserver it is connected to. Parent of a non-hub switch viewserver is the hub switch viewserver in its department. Parent of an end-system is the non-hub switch viewserver it is connected to.

We use only one address for each end-system. The viewserver-address of an end-system is the concatenation of four ids. Thus, the address of s is $z.v.u.s$. Similarly, the address of d is $z.v.x.d$. To obtain a route between s and d, it suffices to obtain views of viewservers u, v, x.

The second viewserver scheme is referred to as **base-QT** (where the QT stands for "query up to top"). It is identical to *base* except that during the query protocol all the viewservers in the source and the destination addresses are queried. That is, to obtain a route between s and d, the views of u, v, x, z are obtained.

The third viewserver scheme is referred to as **vertex-extension**. It is identical to *base* except that viewserver precincts are extended as follows: Let P denote the precinct of a viewserver in the *base* scheme. For each node u in P, if there is a link from node u to node v and v is not in P, node v is added to the precinct; among v's links, only the ones to nodes in P are added to the view. In the example, nodes z, y, x, q are added to the precinct of u, but outgoing links of these nodes to other nodes are not included (e.g. (x, p) and (z, q) are not included). The advantage of this scheme is that even though it increases the precinct size by a factor of d (where d is the average number of neighbors to a node), it increases the number of links stored in the view by a factor less than 2.

The fourth viewserver scheme is referred to as **vertex-extension-QT**. It is identical to *vertex-extension* except that during the query protocol all the viewservers in the source and the destination addresses are queried.

6 Numerical Results

6.1 Results for Network 1

The parameters of the first network topology, referred to as Network 1, are given in Section 5. The link capacity C is taken to be 20 [6], i.e. a link is capable of carrying 20 VCs simultaneously.

Our evaluation measures were computed for a (randomly chosen but fixed) set of 100,000 VC setup requests. Table 1 lists for each viewserver scheme (1) the minimum, average and maximum of the precinct sizes (in number of nodes), (2) the minimum, average and maximum of the merged view sizes (in number of nodes), and (3) the minimum, average and maximum of the number of viewservers queried.

Scheme	Precinct Size	Merged View Size	No. of Viewservers Queried
base	5 / 16.32 / 28	4 / 56.46 / 81	1 / 5.49 / 6
base-QT	5 / 16.32 / 28	27 / 59.96 / 81	6 / 6.00 / 6
vertex-extension	22 / 88.11 / 288	14 / 155.86 / 199	1 / 5.49 / 6
vertex-extension-QT	22 / 88.11 / 288	113 / 163.28 / 199	6 / 6.00 / 6

Table 1: Precinct sizes, merged view sizes, and number of viewservers queried for Network 1.

The precinct size indicates the memory requirement at a viewserver. More precisely, the memory requirement at a viewserver is O(precinct size × d), except for the *vertex-extension* and *vertex-extension-QT* schemes. In these schemes, the memory requirement is increased by a factor less than two. Hence these schemes have the same order of viewserver memory requirement as the *base* and *base-QT* schemes.

The merged view size indicates the memory requirement at a source end-system during the query protocol; i.e. the memory requirement at a source end-system is O(merged view size × d) except for the *vertex-extension* and *vertex-extension-QT* schemes. Note that the source end-system does not need to store information about end-systems other than itself and the destination. The numbers in Table 1 take advantage of this.

The number of viewservers queried indicates the communication time required to obtain the merged view at the source end-system. Hence, the "real-time" communication time required to obtain the merged view at a source is slightly more than one round-trip time between the source and the destination.

As is apparent from Table 1, using a *QT* scheme increases the merged view size by about 6%, and the number of viewservers queried by about 9%. Using the *vertex-extension* scheme increases the merged view size by about 3 times (note that the amount of actual memory needed increases only by a factor less than 2).

The above measures show the memory and time requirements of our protocols. They clearly indicate the savings in storage over the simple approach as manifested by the smaller view sizes. To answer whether the viewserver hierarchy finds many feasible paths, other evaluation measures such as the carried VC load and the percent VC blocking are of interest. They are defined as follows:

- *Carried VC load* is the average number of VCs carried by the network.
- *Percent VC blocking* is the percentage of VC setup requests that are blocked due to the fact that a feasible path is not found.[7]

In our experiments, we keep the average VC lifetime ($1/\mu$) fixed at 15000 and vary the arrival rate of VC setup requests (λ). Figure 8 shows the carried VC load versus λ for the simple approach and the viewserver schemes. Figure 9 shows the percent VC blocking versus λ. At low values of λ, all the viewserver schemes are very close to the simple approach. At moderate values of λ, the *base* and *base-QT* schemes perform badly. The *vertex-extension* and *vertex-extension-QT* schemes are still very close to the simple approach (only 3.4% less carried VC load). Note that the performance of the viewserver schemes can be further improved by trying more viewserver addresses.

Surprisingly, at high values of λ, all the viewserver schemes perform better than the simple approach. At $\lambda = 0.5$, the network with the *base* scheme carries about 30% higher load than the simple approach. This is an interesting result. Our explanation is as follows. Elsewhere [2], we have found that when the viewserver schemes can not find an existing feasible path, this path is usually very long (more than 11 hops). This causes our viewserver hierarchy protocols to reject VCs that are admitted by the simple approach over long paths. The use of long paths for VCs is undesirable since it ties up resources at more intermediate nodes, which can be used to admit many shorter length VCs.

In conclusion, we recommend the *vertex-extension* scheme as it performs close to or better than all other schemes in terms of VC carried load and blocking probability over a wide range of workload. Note that for all viewserver schemes, adding *QT* yields slightly further improvement.

[7] Recall that we assume a blocked VC setup request is cleared (i.e. lost).

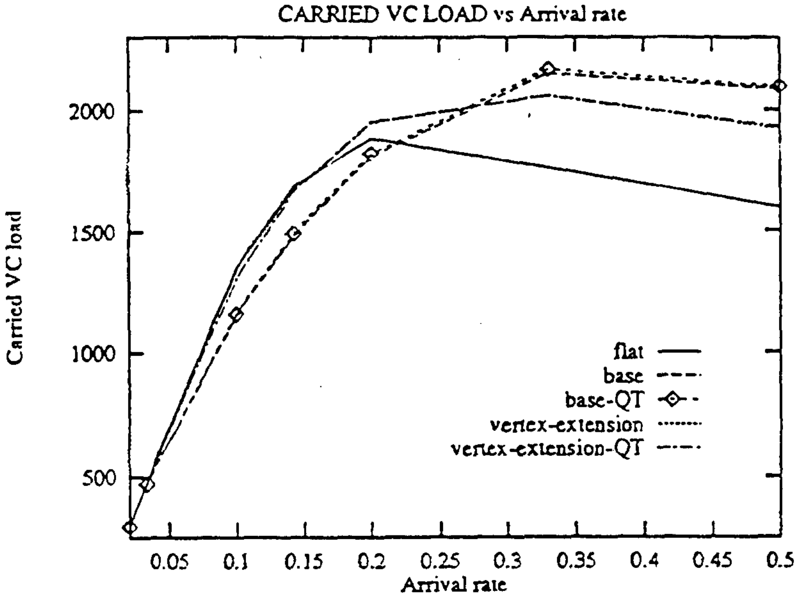


Figure 8: Carried VC load versus arrival rate for Network 1.

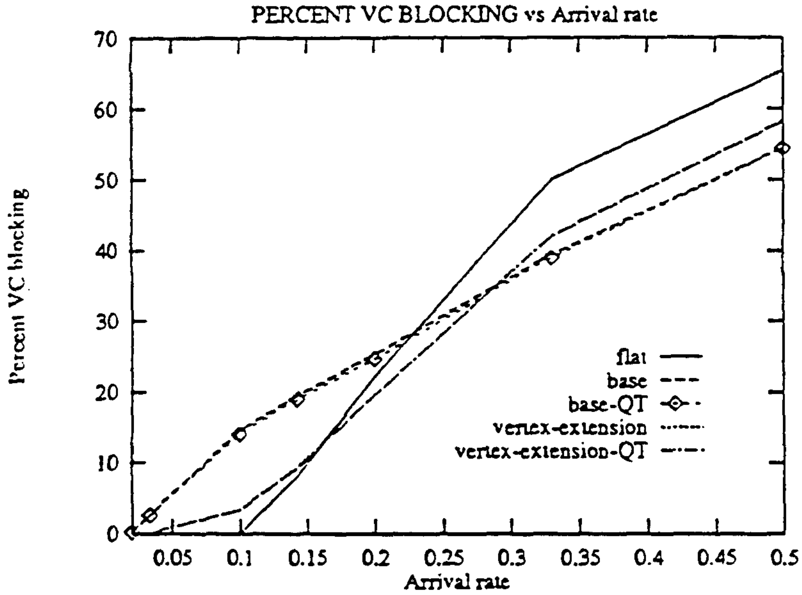


Figure 9: Percent VC blocking versus arrival rate for Network 1.

6.2 Results for Network 2

The parameters of the second network, referred to as Network 2, are the same as the parameters of Network 1. However, a different seed is used for the random number generation, resulting in a different topology and distribution of source-destination end-system pairs for the VCs.

We again take $C = 20$, and we fix $1/\mu$ at 15000. Our evaluation measures were computed for a set of 100,000 VC setup requests. Table 2, and Figures 10 and 11 show the results. Similar conclusions to Network 1 hold for Network 2. An interesting exception is that at high values of λ, we observe that the *vertex-extension* scheme performs slightly better than the *vertex-extension-QT* scheme (about 4.2% higher carried VC load). The reason is the following: Adding QT gives richer merged views, and hence increases the chance of finding a feasible path that is possibly long. As explained in Section 6.1, this results in performance degradation.

Scheme	Precinct Size	Merged View Size	No. of Viewservers Queried
base	4 / 16.32 / 33	4 / 57.61 / 80	1 / 5.52 / 6
base-QT	4 / 16.32 / 33	30 / 60.64 / 80	6 / 6.00 / 6
vertex-extension	17/ 90.36 / 282	16 / 159.70 / 214	1 / 5.52 / 6
vertex-extension-QT	17 /90.36 / 282	113 / 166.97 / 214	6 / 6.00 / 6

Table 2: Precinct sizes, merged view sizes, and number of viewservers queried for Network 2.

We have repeated the above evaluations for other networks and obtained similar conclusions.

7 Conclusions

We presented a hierarchical VC routing protocol for ATM-like networks. Our protocol satisfies QoS constraints, adapts to dynamic topology changes, and scales well to large number of nodes.

Our protocol uses partial views maintained by viewservers. The viewservers are organized hierarchically. To setup a VC, the source end-system queries viewservers to obtain a merged view that contains itself and the destination end-system. This merged view is then used to compute a source route for the VC.

We evaluated several viewserver hierarchy schemes and compared them to the simple approach. Our results on 2764-node networks indicate that the *vertex-extension* scheme performs close to or better than the simple approach in terms of VC carried load and blocking probability over a wide range of real-time workload. It also reduces the amount of memory requirement by up to two order of magnitude. We note that our protocol scales even better on larger size networks [3].

In all the viewserver schemes we studied, each switch is a viewserver. In practice, not all switches need to be viewservers. We may associate one viewserver with a group of switches. This is particularly attractive in ATM networks where each signaling entity is responsible for establishing VCs across a group of nodes. In such an environment, viewservers and signaling entities can be

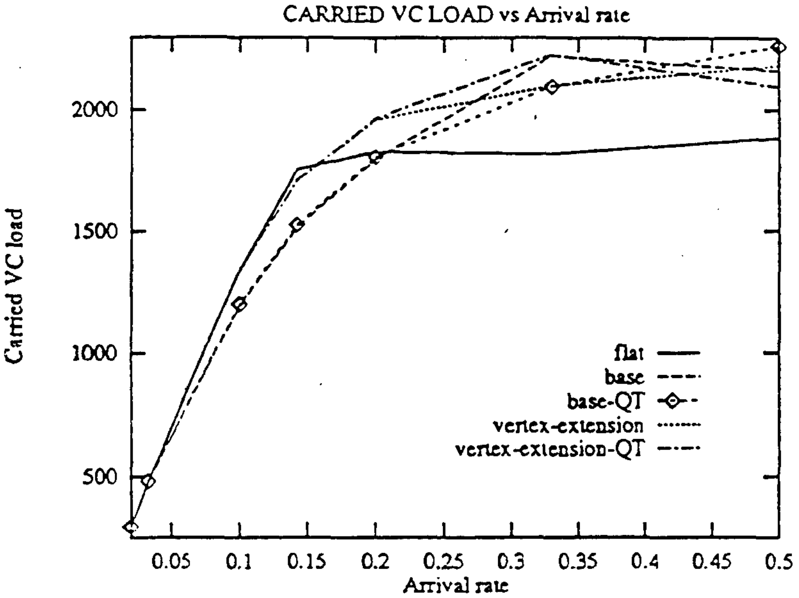


Figure 10: Carried VC load versus arrival rate for Network 2.

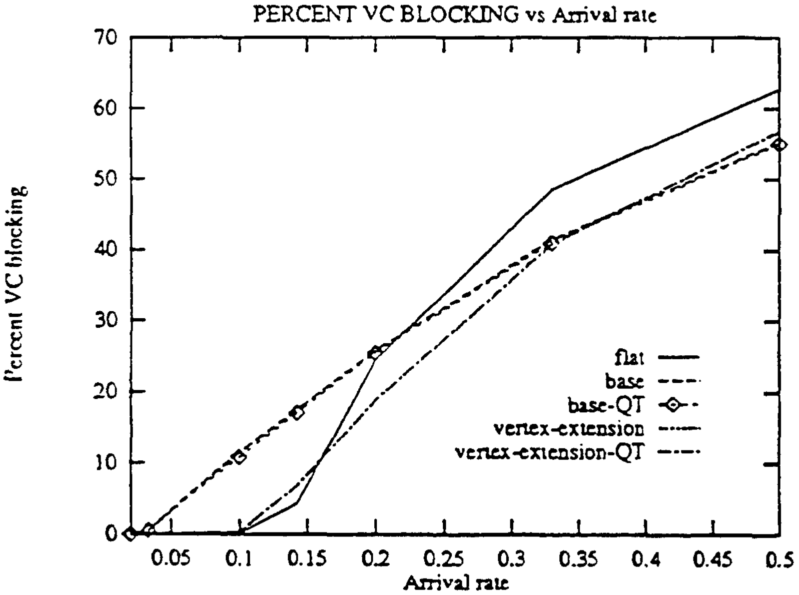


Figure 11: Percent VC blocking versus arrival rate for Network 2.

combined.

However, there is an advantage of each switch being a viewserver; that is, source nodes do not require fixed source routes to their parent viewservers (in the view-query protocol). This reduces the amount of hand configuration required. In fact, the *base* and *base-QT* viewserver schemes do not require any hand configuration.

Our evaluation model assumed that views are instantaneously updated, i.e. no delayed feedback between link cost changes and view/route changes. We plan to investigate the effect of delayed feedback on the performance of the different schemes. We expect our viewserver schemes to outperform the simple approach in this realistic setting as the update of views of the viewservers requires less time and communication overhead. Thus, views in our viewserver schemes will be more up-to-date.

As we pointed out in [3], the only drawback of our protocol is that to obtain a source route for a VC, views are merged at (or prior to) the VC setup, thereby increasing the setup time. This drawback is not unique to our scheme [8, 16, 7, 11]. Reference [3] describes several ways, including cacheing and replication, to reduce the setup overhead and improve performance.

References

[1] H. Ahmadi, J. Chen, and R. Guerin. Dynamic Routing and Call Control in High-Speed Integrated Networks. In Proc. *Workshop on Systems Engineering and Traffic Engineering, ITC'13*, pages 19–26, Copenhagen, Denmark, June 1991.

[2] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A New Inter-Domain Routing Protocol and its Evaluation. Technical Report UMIACS-TR-93-98, CS-TR-3151, Department of Computer Science, University of Maryland, College Park, October 1993. Earlier version CS-TR-3033, February 1993.

[3] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A New Inter-Domain Routing Protocol. In Proc. *IEEE INFOCOM '94*, Toronto, Canada, June 1994.

[4] A. Alles. ATM in Private Networking: A Tutorial. Hughes LAN Systems, 1993.

[5] P. Almquist. Type of Service in the Internet Protocol Suite. Technical Report RFC-1349, Network Working Group, July 1992.

[6] L. Breslau, D. Estrin, and L. Zhang. A Simulation Study of Adaptive Source Routing in Integrated Services Networks. Available by anonymous ftp at catarina.usc.edu:pub/breslau, September 1993.

[7] J. N. Chiappa. A New IP Routing and Addressing Architecture. Big-Internet mailing list., 1992. Available by anonymous ftp from `munnari.oz.au:big-internet/list-archive`.

[8] D.D. Clark. Policy routing in Internet protocols. Request for Comment RFC-1102, Network Information Center, May 1989.

[9] R. Coltun and M. Sosa. VC Routing Criteria. Internet Draft, March 1993.

[10] D. Comer and R. Yavatkar. FLOWS: Performance Guarantees in Best Effort Delivery Systems. In Proc. *IEEE INFOCOM, Ottawa, Canada*, pages 100–109, April 1989.

[11] D. Estrin, Y. Rekhter, and S. Hotz. Scalable Inter-Domain Routing Architecture. In Proc. *ACM SIGCOMM '92*, pages 40–52, Baltimore, Maryland, August 1992.

[12] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent Capacity and its Application to Bandwidth Allocation in High-Speed Networks. *IEEE J. Select. Areas Commun.*, SAC-9(7):968–981, September 1991.

[13] A. Guillen, R. Kia, and B. Sales. An Architecture for Virtual Circuit/QoS Routing. In Proc. *IEEE International Conference on Network Protocols '93*, pages 80–87, San Francisco, California, October 1993.

[14] S. Gupta, K. Ross, and M. ElZarki. Routing in Virtual Path Based ATM Networks. In Proc. *GLOBECOM '92*, pages 571–575, 1992.

[15] R-H. Hwang, J. Kurose, and D. Towsley. MDP Routing in ATM Networks Using Virtual Path Concept. In Proc. *IEEE INFOCOM*, pages 1509–1517, Toronto, Ontario, Canada, June 1994.

[16] M. Lepp and M. Steenstrup. An Architecture for Inter-Domain Policy Routing. Internet Draft. Available from the authors., June 1992.

[17] I. Matta and A.U. Shankar. An Iterative Approach to Comprehensive Performance Evaluation of Integrated Services Networks. In Proc. *IEEE International Conference on Network Protocols '94*, Boston, Massachusetts, October 1994. To appear.

[18] J. Moy. OSPF Version 2. RFC 1247, Network Information Center, SRI International, July 1991.

[19] C. Parris and D. Ferrari. A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks. Technical Report TR-93-005, International Computer Science Institute, Berkeley, California, January 1993.

[20] J. Postel. Transmission Control Protocol: DARPA Internet Program Protocol Specification. Request for Comment RFC-793, Network Information Center, SRI International, 1981.

[21] M. Prycker. *Asynchronous Transfer Mode - Solution for Broadband ISDN.* Ellis Horwood, 1991.

[22] S. Rampal, D. Reeves, and D. Agrawal. An Evaluation of Routing and Admission Control Algorithms for Multimedia Traffic in Packet-Switched Networks. Available from the authors, 1994.

[23] H. Suzuki and F. Tobagi. Fast Bandwidth Reservation Scheme with Multi-Link and Multi-Path Routing in ATM Networks. In Proc. *IEEE INFOCOM '92*, pages 2233–2240, Florence, Italy, May 1992.

[24] E. Sykas, K. Vlakos, I. Venieris, and E. Protonotarios. Simulative Analysis of Optimal Resource Allocation and Routing in IBCN's. *IEEE J. Select. Areas Commun.*, 9(3):486–492, April 1991.

[25] P. F. Tsuchiya. The Landmark Hierarchy: Description and Analysis, The Landmark Routing: Architecture Algorithms and Issues. Technical Report MTR-87W00152, MTR-87W00174, The MITRE Corporation, McLean, Virginia, 1987.

[26] P. F. Tsuchiya. The Landmark Hierarchy:A New Hierarchy For Routing In Very Large Networks. In Proc. *ACM SIGCOMM '88*, August 1988.

REPORT DOCUMENTATION PAGE

OMB No 0704-0188

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
October 1994

3. REPORT TYPE AND DATES COVERED
Technical

4. TITLE AND SUBTITLE
A Scalable Virtual Circuit Routing Scheme for ATM Networks

5. FUNDING NUMBERS
C DASG60-92-0055
G NCR 89-04590
G NCR 93-21043

6. AUTHOR(S)
Cengiz Alaettinoglu, Ibrahim Matta and A. Udaya Shankar

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Computer Science
A. V. Willliams Building
University of Maryland
College Park, MD 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
CSTR-3360
UMIACS-TR 94-115

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Phillips Laboratory
Directorate of Contracting
3651 Lowry Avenue SE
Kirtland AFB NM 87117-5777

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

High-speed networks, such as ATM networks, are expected to support diverse quality-of-service (QoS) requirements, including real-time QoS. Real-time QoS is required by many applications such as voice and video. To support such service, routing protocols based on the Virtual Circuit (VC) model have been proposed. However, these protocols do not scale well to large networks in terms of storage and communication overhead.

In this paper, we present a scalable VC routing protocol. It is based on the recently proposed viewserver hierarchy, where each viewserver maintains a partial view of the network. By querying these viewservers, a source can obtain a merged view that contains a path to the destination. The source then sends a request packet over this path to setup a real-time VC through resource reservations. The request is blocked if the setup fails. We compare our protocol to a simple approach using simulation. Under this simple approach , a source maintains a full view of the network. In addition to the savings in storage, our results indicate that our protocol performs close or better than the simple approach in terms of VC carried load and blocking probability over a wide range of real-time workload.

14. SUBJECT TERMS
Computer-Communication Networks: Network Architecture and Design, Network Protocols;Routing Protocols: Computer Network Routing Protocols

15. NUMBER OF PAGES
22 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Hierarchical Inter-Domain Routing Protocol with On-Demand ToS and Policy Resolution*

Cengiz Alaettinoğlu, A. Udaya Shankar

Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, Maryland 20742

June 20, 1994

Abstract

Traditional inter-domain routing protocols based on superdomains maintain either "strong" or "weak" ToS and policy constraints for each visible superdomain. With strong constraints, a valid path may not be found even though one exists. With weak constraints, an invalid domain-level path may be treated as a valid path.

We present an inter-domain routing protocol based on superdomains, which always finds a valid path if one exists. Both strong and weak constraints are maintained for each visible superdomain. If the strong constraints of the superdomains on a path are satisfied, then the path is valid. If only the weak constraints are satisfied for some superdomains on the path, the source uses a query protocol to obtain a more detailed "internal" view of these superdomains, and searches again for a valid path. Our protocol handles topology changes, including node/link failures that partition superdomains. Evaluation results indicate our protocol scales well to large internetworks.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*packet networks; store and forward networks*; C.2.2 [Computer-Communication Networks]: Network Protocols—*protocol architecture*; C.2.m [Routing Protocols]; F.2.m [Computer Network Routing Protocols].

* This work is supported in part by ARPA and Philips Labs under contract DASG60-92-0055 to Department of Computer Science, University of Maryland, and by National Science Foundation Grant No. NCR 89-04590. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, PL, NSF, or the U.S. Government. Computer facilities were provided in part by NSF grant CCR-8811954.

1 Introduction

A computer internetwork, such as the Internet, is an interconnection of backbone networks, regional networks, metropolitan area networks, and stub networks (campus networks, office networks and other small networks)[1]. Stub networks are the producers and consumers of the internetwork traffic, while backbones, regionals and MANs are transit networks. Most of the networks in an internetwork are stub networks. Each network consists of nodes (hosts, routers) and links. A node that has a link to a node in another network is called a *gateway*. Two networks are *neighbors* when there is one or more links between gateways in the two networks (see Figure 1).

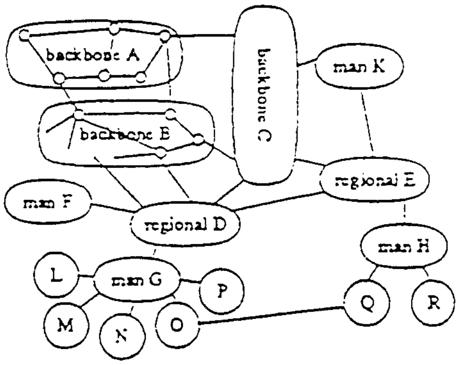


Figure 1: A portion of an internetwork. (Circles represent stub networks.)

An internetwork is organized into *domains*[2]. A domain is a set of networks (possibly consisting of only one network) administered by the same agency. Domains are typically subject to *policy constraints*, which are administrative restrictions on inter-domain traffic [7, 11, 8, 5]. The policy constraints of a domain U are of two types: *transit policies*, which specify how other domains can use the resources of U (e.g. \$0.01 per packet, no traffic from domain V); and *source policies*, which specify constraints on traffic originating from U (e.g. domains to avoid/prefer, acceptable connection cost). Transit policies of a domain are public (i.e. available to other domains), whereas source policies are usually private.

Within each domain, an *intra-domain routing protocol* is executed that provides routes between source and destination nodes in the domain. This protocol can be any of the typical ones, i.e., next-hop or source routes computed using distance-vector or link-state algorithms. To satisfy

[1] For example, NSFNET, MILNET are backbones, and Suranet, CerfNet are regionals.

[2] Also referred to as *routing domains* or *administrative domains*.

type-of-service (ToS) constraints of applications (e.g. low delay, high throughput, high reliability, minimum monetary cost), each node maintains a *cost* for each outgoing link and ToS. The intra-domain routing protocol should choose optimal paths based on these costs.

Across all domains, an *inter-domain routing protocol* is executed that provides routes between source and destination nodes in different domains, using the services of the intra-domain routing protocols within domains. This protocol should have the following properties:

(1) It should satisfy the policy constraints of domains. To do this, it must keep track of the policy constraints of domains [5].

(2) An inter-domain routing protocol should also satisfy ToS constraints of applications. To do this, it must keep track of the ToS services offered by domains [5].

(3) An inter-domain routing protocol should scale up to very large internetworks, i.e. with a very large number of domains. Practically this means that processing, memory and communication requirements should be *much less than linear* in the number of domains. It should also handle non-hierarchical domain interconnections at any level [8] (e.g. we do not want to hand-configure special routes as "back-doors").

(4) An inter-domain routing protocol should automatically adapt to link cost changes and node/link failures and repairs, including failures that partition domains [13].

A Straight-Forward Approach

A straight-forward approach to inter-domain routing is domain-level source routing with link-state approach [7, 5]. In this approach, each router[3] maintains a *domain-level view* of the internetwork, i.e., a graph with a vertex for every domain and an edge between every two neighbor domains. Policy and ToS information is attached to the vertices and the edges of the view.

When a source node needs to reach a destination node, it (or a router[4] in the source's domain) first examines this view and determines a *domain-level source route* satisfying ToS and policy constraints, i.e., a sequence of domain ids starting from the source's domain and ending with the destination's domain. Then packets are routed to the destination using this domain-level source route and the intra-domain routing protocols of the domains crossed.

For example, consider the internetwork of Figure 2 (each circle is a domain, and each thin line

[3] Not all nodes maintain routing tables. A router is a node that maintains a routing table.

[4] referred to as the policy server in [7]

is a domain-level interconnection). Suppose a node in $d1$ desires a connection to a node in $d7$. Suppose the policy constraints of $d3$ and $d19$ do not allow transit traffic originating from $d1$. Every node maintains this information in its view. Thus the source node can choose a valid path from source domain $d1$ to destination domain $d7$ avoiding $d3$ and $d19$ (e.g. thick line in the figure).

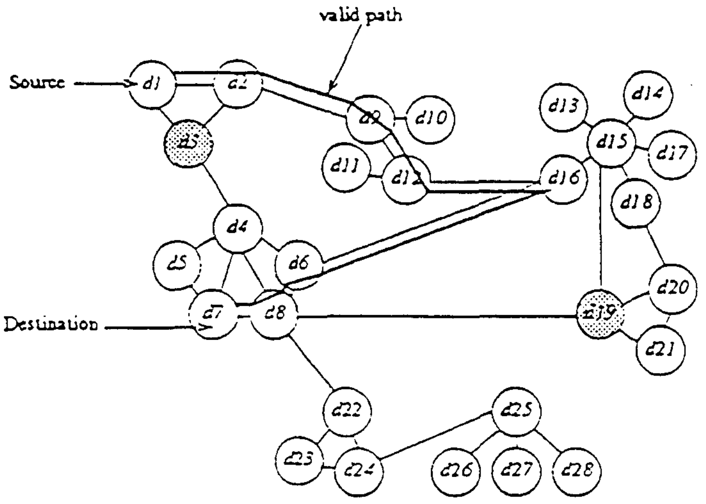


Figure 2: An example interdomain topology.

The disadvantage of this straightforward scheme is that it does not scale up for large internetworks. The storage at each router is proportional to $N_D \times E_D$, where N_D is the number of domains and E_D is the average number of neighbor domains to a domain. The communication cost for updating views is proportional to $N_R \times E_R$, where N_R is the number of routers in the internetwork and E_R is the average router neighbors of a router (topology changes are flooded to all routers in the internetwork).

The Superdomain Approach

To achieve scaling, several approaches based on hierarchically aggregating domains into *superdomains* have been proposed [16, 14, 6]. Here, each domain is a level 1 superdomain, "close" level 1 superdomains are grouped into level 2 superdomains, "close" level 2 superdomains are grouped into level 3 superdomains, and so on (see Figure 3). Each router x maintains a view that contains the level 1 superdomains in x's level 2 superdomain, the level 2 superdomains in x's level 3 superdomain (excluding the x's level 2 superdomain), and so on. Thus a router maintains a smaller view than it would in the absence of hierarchy. For the superdomain hierarchy of Figure 3, the views of two

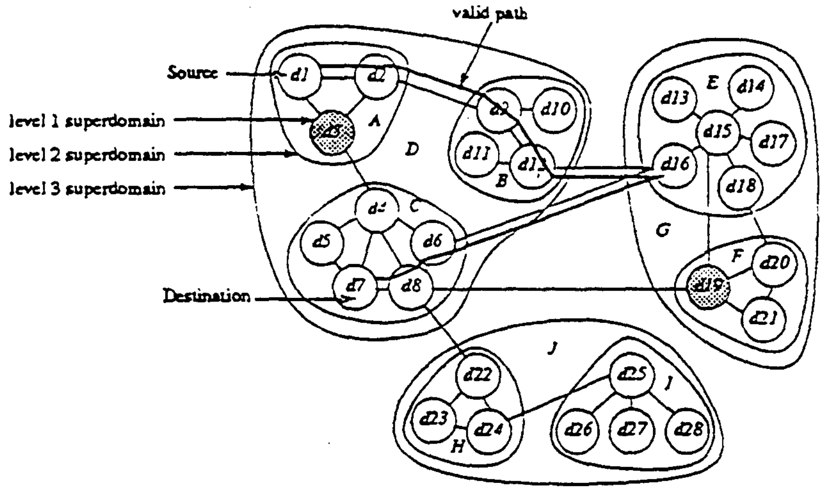


Figure 3: An example of superdomain hierarchy.

routers (one in domain $d1$ and one in domain $d16$) are shown in Figures 4 and 5.

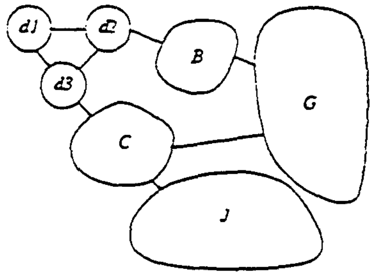


Figure 4: View of a router in $d1$.

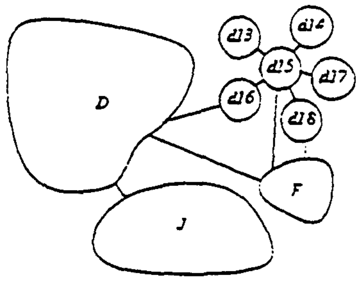


Figure 5: View of a ~~router~~ in $d16$.

The superdomain approach has several problems. One problem is that the aggregation results in loss of domain-level ToS and policy information. A superdomain is usually characterized by a single set of ToS and policy constraints derived from the ToS and policy constraints of the domains in it. Routers outside the superdomain assume that this set of constraints applies uniformly to each of its children (and by recursion to each domain in the superdomain). If there are domains with different (possibly contradictory) constraints in a superdomain, then there is no good way of deriving the ToS and policy constraints of the superdomain.

The usual technique [16] of obtaining ToS and policy constraints of a superdomain is to obtain either a *strong* set of constraints or a *weak* set of constraints[5] from the ToS and policy constraints of

[5] "strong" and "weak" are referred to respectively as "union" and "intersection" in [16]

the children superdomains in it. If strong (weak) constraints are used for policies, the superdomain enforces a policy constraint if that policy constraint is enforced by some (all) of its children. If strong (weak) constraints are used for ToS constraints, the superdomain is assumed to support a ToS if that ToS is supported by all (some) of its children. The intention is that if strong (weak) constraints of a superdomain are (are not) satisfied then any (no) path through that superdomain is valid.

Each approach has problems. Strong constraints can eliminate valid paths, and weak constraints can allow invalid paths. For example in Figure 3, $d16$ allows transit traffic from $d1$ while $d19$ does not; with strong constraints G would not allow transit traffic from $d1$, and with weak constraints G would allow transit traffic from $d1$ to be routed via $d19$.

Other problems of the superdomain approach are that the varying visibilities of routers complicates superdomain-level source routing and handling of node/link failures (especially those that partition superdomains). The usual technique for solving these problems is to augment superdomain-level views with gateways [16] (see Section 3).

Our Contribution

In this paper, we present an inter-domain routing protocol based on superdomains, which finds a valid path if and only if one exists. Both strong and weak constraints are maintained for each visible superdomain. If the strong constraints of the superdomains on a path are satisfied, then the path is valid. If only the weak constraints are satisfied for some superdomains on the path, the source uses a query protocol to obtain a more detailed "internal" view of these superdomains, and searches again for a valid path.

We use superdomain-level views with gateways and a link-state view update protocol to handle topology changes including failures that partition superdomains. The storage cost is $O(\log N_D \times \log N_D)$ without the query protocol. We demonstrate the scaling properties of the query protocol by giving evaluation results based on simulations. Our evaluation results indicate that the query protocol can be performed using 15% extra space.

Our protocol consists of two subprotocols: a **view-query protocol** for obtaining views of greater resolution when needed; and a **view-update protocol** for disseminating topology changes to the views.

Several approaches to scalable inter-domain routing have been proposed, based on the superdomain hierarchy [1, 14, 16, 9, 6], and the landmark hierarchy [18, 17]. Some of these approaches suffer from loss of ToS and policy information (and hence may not find a valid path which exists). Others are still in a preliminary stage. (Details in Section 8.)

One important difference between these approaches and ours is that ours uses a query mechanism to obtain ToS and policy details whenever needed. In our opinion, such a mechanism is needed to obtain a scalable solution. Query protocols are also being developed to enhance the protocols in [9, 6]. Reference [2] presents protocols based on a new kind of hierarchy, referred to as the viewserver hierarchy (more details in Section 8).

A preliminary version of the view-query protocol was proposed in reference [1]. That version differs greatly from the one in this paper. Here, we augment superdomain-level views with gateways. In [1], we augmented superdomain-level views with superdomain-to-domain edges (details in Section 8). Both versions have the same time and space complexity, but the protocols in this paper are much simpler conceptually. Also the view-update protocol is not in reference [1].

Organization of the paper

In Section 2, we present some definitions used in this paper. In Section 3, we define the view data structures. In Section 4, we describe how views are affected by topology changes. In Section 5, we present the view-query protocol. In Section 6, we present the view-update protocol. In Section 7, we present our evaluation model and the results of its application to the superdomain hierarchy. In Section 8, we survey recent approaches to inter-domain routing. In Section 9, we conclude and describe cacheing and heuristic schemes to improve performance.

2 Preliminaries

Each domain has a unique id. Let `DomainIds` denote the set of domain-ids. Each node has a unique id. Let `NodeIds` denote the set of node-ids. For a node x, we use domainid(x) to denote the domain-id of x's domain.

The superdomain hierarchy defines the following parent-child relationship: a level i, $i > 1$, superdomain is the parent of each level $i - 1$ superdomain it contains. Top-level superdomains have no parents. Level 1 superdomains, which are just domains, have no children. For any two superdomains X and Y, X is a sibling of Y iff X and Y have the same parent. X is an ancestor (descendant) of Y iff $X = Y$ or X is an ancestor (descendant) of Y's parent (child).

Each router maintains information about a subset of superdomains, referred to as its visible superdomains. The *visible superdomains* of a router x are (1) x's domain itself, (2) siblings of x's domain, and (3) siblings of ancestors of x's domain. In Figure 3, the visible superdomains of a router in $d1$ are $d1, d2, d3, B, C, G, J$ (these are shown in Figure 4). Note that if a superdomain U is visible to a router, then no ancestor or descendant of U is visible to the router.

Each superdomain has a unique id, i.e. unique among all superdomains regardless of level. Let SuperDomainIds denote the set of superdomain-ids. DomainIds is a subset of SuperDomainIds. For a superdomain U, let level(U) denote the level of U in the hierarchy, let Ancestors(U) denote the set of ids of ancestor superdomains of U in the hierarchy, and let Children(U) denote the set of ids of child superdomains of U in the hierarchy.

For a router x, let VisibleSuperDomains(x) denote the set of ids of superdomains visible from x.

We extend the above definitions by allowing their arguments to be nodes, in which case the node stands for its domain. For example, if x is a node in domain d, Ancestors(x) denotes Ancestors(d).

3 Superdomain-Level Views with Gateways

For routing purposes, each domain (and node) has an address, defined as the concatenation of the superdomain ids starting from the top level and going down to the domain (node). For example in Figure 3, the address of domain $d15$ is $G.E.d15$, and the address of a node h in $d15$ is $G.E.d15.h$.

When a source node needs to reach a destination node, it first determines the visible superdomain in the destination address and then by examining its view determines a superdomain-level source route (satisfying ToS and policy constraints) to this superdomain. However, since routers in different superdomains maintain views of different sets of superdomains, this superdomain-level source route can be meaningless at some intermediate superdomain's router x because the next superdomain in this source route is not visible to x. For example in Figure 4, superdomain-level source route $\langle d2, B, G, C\rangle$ created at a router in $d2$ becomes meaningless once the packet is in G, where C is not visible.

The usual technique of solving this problem is to augment superdomain-level views with gateways and edges between these gateways.

Define the pair $U{:}g$ to be an *sd-gateway* iff U is a superdomain and g is a node that is in U and has a link to a node outside U. Equivalently, we say that g is *a gateway of* U.

Define $\langle U{:}g, h\rangle$ to be an *actual-edge* iff $U{:}g$ is an sd-gateway, h is a gateway not in U, and there is a link from g to h.

Define $\langle U{:}g, h\rangle$ to be a *virtual-edge* iff $U{:}g$ and $U{:}h$ are sd-gateways and $g \neq h$ (note that there may not be a link between g and h).

$\langle U{:}g, h\rangle$ is an *edge* iff it is an actual-edge or a virtual-edge. An edge $\langle U{:}g, h\rangle$ is also said to be an *outgoing edge* of $U{:}g$. Define *edges of* $U{:}g$ to be the set of edges outgoing from $U{:}g$. Define *edges of* U to be the set of edges outgoing from any gateway of U.

Let Gateways(U) denote the set of node-ids of gateways of U. Let Edges($U{:}g$) denote the edges of $U{:}g$. Note that we never use "edge" as a synonym for link.

A gateway g of a domain can generate many sd-gateways, specifically, $U{:}g$ for every ancestor U of g's domain such that g has a link to a node outside U. A link $\langle g, h\rangle$ where g and h are gateways in different domains, can generate many actual-edges; specifically, actual-edge $\langle U{:}g, h\rangle$ for every ancestor U of g's domain such that U is not an ancestor of h's domain.

For the internetwork topology of Figure 2, the corresponding gateway-level connections are shown in Figure 6 where black rectangles are gateways. For the hierarchy of Figure 3, gateway g in Figure 6 generates sd-gateways $d16{:}g$, $E{:}g$, and $G{:}g$. The link $\langle g, h\rangle$ in Figure 6 generates actual-edges $\langle d16{:}g, h\rangle$, $\langle E{:}g, h\rangle$, $\langle G{:}g, h\rangle$.

To a router, at most one of the sd-gateways generated by a gateway g is visible, namely $U{:}g$ where U is an ancestor of g's domain and U is visible to the router. At most one of the actual-edges generated by a link $\langle g, h\rangle$ between two gateways in different domains is visible to the router, namely edge $\langle U{:}g, h\rangle$ where $U{:}g$ is visible to the router. None of the actual-edges are visible to the router if g and h are inside a visible superdomain. For example in Figure 3, of the actual-edges generated by link $\langle g, h\rangle$, only $\langle G{:}g, h\rangle$ is visible to a router in $d1$, and only $\langle d16{:}g, h\rangle$ is visible to a router in $d16$.

A router maintains a view consisting of the visible sd-gateways and their outgoing actual- and virtual-edges. An edge $\langle U{:}g, h\rangle$ in the view of a router connects the sd-gateway $U{:}g$ to the sd-

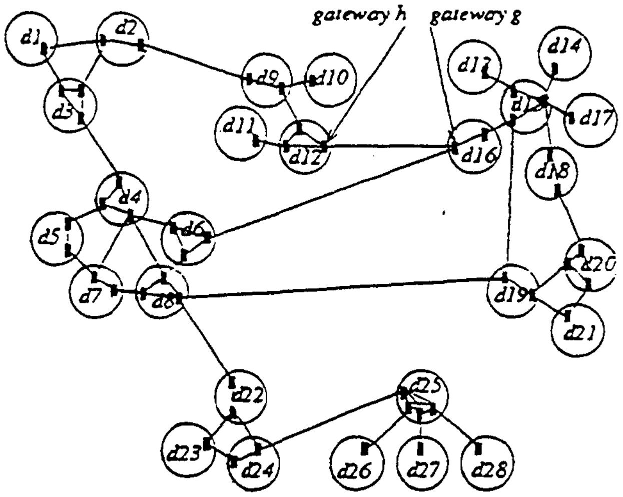


Figure 6: Gateway-level connections of internetwork of Figure 2.

gateway $V{:}h$ such that $V{:}h$ is visible to the router. For the superdomain-level views of Figures 4 and 5, the new views are shown in Figures 7 and 8, respectively.

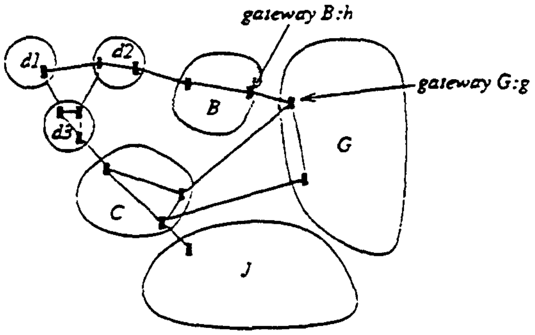


Figure 7: View of a router in $d1$.

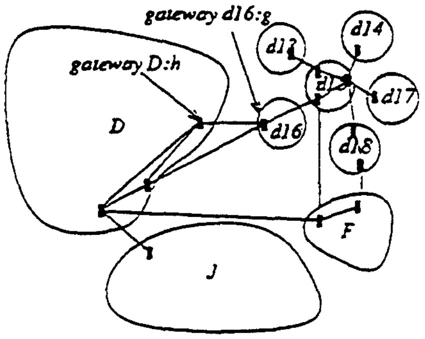


Figure 8: View of a router in $d16$.

The view of a router x contains, for each superdomain U that is visible to x or is an ancestor of x, the strong and weak constraints of U and a set referred to as $\mathit{Gateways\&Edges}_x(U)$. This set contains, for each gateway y of U, the edges of $U{:}y$ and their costs. The reason for storing information about ancestor superdomains is given in Section 5. The cost field is used to satisfy ToS constraints and is described in Section 4. The *timestamp* field is described in Section 6. Formally, the view of x is defined as follows:

$View_x$. View of x.

$= \{\langle U,\ \texttt{strong\_constraints}(U),\ \texttt{weak\_constraints}(U),\ \mathit{Gateways\&Edges}_x(U)\rangle :$

$U \in \texttt{VisibleSuperDomains}(x) \cup \texttt{Ancestors}(x)\ \}$

where

$\mathit{Gateways\&Edges}_x(U)$. Sd-gateways and edges of U.

$= \{\langle y,\ \mathit{timestamp},\ \{\langle z,\ \mathit{cost}\rangle :\ \langle U{:}y, z\rangle \in \texttt{Edges}(U{:}y)\}\rangle :\ y \in \texttt{Gateways}(U)\ \}$.

ToS and policy constraints can also be specified for each sd-gateway and edge. Our protocols can be extended to handle such constraints, but we have not done so here in order to keep their descriptions simple.

A *superdomain-level source route* is now a sequence of sd-gateway ids. With this definition, it is easy to verify that whenever the next superdomain in a superdomain-level source route is not visible to a router, there is an actual-edge (hence a link) between the router and the next gateway in this route.

4 Edge-Costs and Topology Changes

A cost is associated with each edge. The cost of an edge equals a vector of values if the edge is up; each cost value indicates how expensive it is to cross the edge according to some ToS constraint. The cost equals ∞ if the edge is an actual-edge and it is down, or the edge is a virtual-edge $\langle U{:}g, h\rangle$ and h can not be reached from g without leaving U.

Since an actual-edge represents a physical link, its cost can be determined from measured link statistics. The cost of a virtual-edge $\langle U{:}g, h\rangle$ is an aggregation of the cost of physical links in U and is calculated as follows: If U is a domain, the cost of $\langle U{:}g, h\rangle$ is calculated as the maximum/minimum/average cost of the routes within U from g to h [4]. For higher level superdomains U, the cost of $\langle U{:}g, h\rangle$ is derived from the costs of edges between the gateways of children superdomains of U.

Link cost changes and link/node failures and repairs correspond to cost changes, failures and repairs of actual- and virtual-edges. Thus the attributes of edges in the views of routers must be regularly updated. For this, we employ a view-update protocol (see Section 6).

Link/node failures can also partition a superdomain into cells, where a *cell* of a superdomain is defined to be a maximal subset of nodes of the superdomain that can reach each other without leaving the superdomain. Superdomain partitions can occur at any level in the hierarchy. For example, suppose U is a domain and V is its parent superdomain. U can be partitioned into cells without V being partitioned (i.e. if the cells of U can reach each other without leaving V). The opposite can also happen: if all links between U and the other children of V fail, then V becomes partitioned but U does not. Or both U and V can be partitioned. In the same way, link/node repairs can merge cells into bigger cells.

We handle superdomain partitioning as follows: A router detects that a superdomain U is partitioned when a virtual-edge of U in the router's view has cost ∞. When a router forwards a packet to a destination for which the visible superdomain, say U, in the destination address is partitioned into cells, a copy of the packet is sent to each cell by sending a copy of the packet to each gateway of U; the id U in the destination address is "marked" in the packet so that subsequent routers do not create new copies of the packet for U.

5 View-Query Protocol

When a source node wants a superdomain-level source route to a destination, a router in its domain examines its view and searches for a valid path (i.e. superdomain-level source route) using the destination address[6]. We refer to this router as the *source router*. Even though the source router does not know the constraints of the individual domains that are to be crossed in each superdomain, it does know the strong and weak constraints of the superdomains. We refer to a superdomain whose strong constraints are satisfied as a *valid superdomain*. If a superdomain's weak constraints are satisfied but strong constraints are not satisfied, then there may be a valid path through this superdomain. We refer to such a superdomain as a *candidate superdomain*.

A path is valid if it involves only valid superdomains. A path cannot be valid if it involves a superdomain which is neither valid nor candidate. We refer to a path involving only valid and candidate superdomains as a *candidate path*.

[6] We assume that the source has the destination's address. If that is not the case, it would first query the name servers to obtain the address for the destination. Querying the name servers can be done the same way it is done currently in the Internet. It requires nodes to have a set of fixed addresses to name servers. This is also sufficient in our case.

If the source router's view contains a candidate path $\langle U_0{:}g_{0_0}, \ldots, U_0{:}g_{0_{n_0}}, U_1{:}g_{1_0}, \ldots, U_1{:}g_{1_{n_1}}, \cdots, U_m{:}g_{m_0}, \ldots, U_m{:}g_{m_{n_m}} \rangle$ to the destination (and does not contain a valid path), then for each candidate superdomain U_i on this path, the source router queries gateway g_{i_0} of U_i for the internal view of U_i. This internal view consists of the constraints, sd-gateways and edges of the child superdomains of U_i.

When a router x receives a request for the internal view of an ancestor superdomain U, it returns the following data structure:

$IView_x(U)$. Internal view of U at router x.

$$= \{\langle V, \texttt{strong\_constraints}(V), \texttt{weak\_constraints}(V), \mathit{Gateways\&Edges}_x(V)\rangle \in View_x : V \in \texttt{Children}(U)\}$$

It is to simplify the construction of $IView_x(U)$ that we store information about ancestor superdomains in the view of router x. Instead of storing this information, router x could construct $IView_x(U)$ from the constraints, sd-gateways and edges of the visible descendants of U. We did not choose this alternative because the extra information does not increase storage complexity.

When the source router receives the internal view of a superdomain U, it does the following: (1) it removes the sd-gateways and edges of U from its view; (2) it adds the sd-gateways and edges of children superdomains in the internal view of U; and (3) searches for a valid path again. If there is still no valid path but there are candidate paths, the process is repeated.

For example, consider Figure 3. For a router in superdomain $d1$ (see Figure 7), G is visible and is a candidate domain. The internal view of G is shown in Figure 9, and the resulting merged view is shown in Figure 10. The valid path through G (visiting $d16$ and avoiding $d19$) can be discovered using this merged view (since the strong constraints of E are satisfied).

Consider a candidate route to a destination: $\langle U_0{:}g_{0_0}, \ldots, U_0{:}g_{0_{n_0}}, U_1{:}g_{1_0}, \ldots, U_1{:}g_{1_{n_1}}, \cdots, U_m{:}g_{m_0}, \ldots, U_m{:}g_{m_{n_m}} \rangle$. If superdomain U_i is partitioned into cells, it may re-appear later in the candidate path (i.e. for some $j \neq i$, $U_j = U_i$). In this case both gateways g_{i_0} and g_{j_0} are queried. Timestamps are used to resolve conflicts between the information reported by these gateways.

The view-query protocol uses two types of messages as follows:

- $(\texttt{RequestIView}, sdid, gid, s\_address, d\_address)$

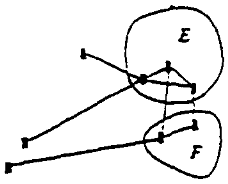


Figure 9: Internal view of G.

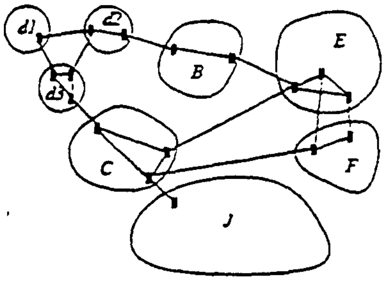


Figure 10: Merged view at $d1$.

Sent by a source router to gateway gid to obtain the internal view of superdomain $sdid$. $s\_address$ is the address of the source router. $d\_address$ is the address of the destination node (of the desired route).

- (ReplyIView, $sdid, gid, iview, d\_address$)

  where $iview$ is the internal view of superdomain $sdid$, and other parameters are as in the RequestIView message. It is sent by gateway gid to the source router.

The state maintained by a source router x is listed in Figure 15. $PendingReq_x$ is used to avoid sending new request messages before receiving all outstanding reply messages. $WView_x$ and $PendingReq_x$ are allocated and deallocated on demand for each destination.

The events of router x are specified in Figure 15. In the figure, * is a wild-card matching any value. $TimeOut_x$ event is executed after a time-out period from the execution of $Request_x$ event to indicate that the request has not been satisfied. The source host can then repeat the same request afterwards.

The procedure $search_x$ uses an operation "ReliableSend(m) to v", where m is the message being sent and v is either an address of an arbitrary router or an id of a gateway of a visible superdomain. ReliableSend is asynchronous. The message is delivered to v as long as there is a sequence of up links between u and v.[7] (Note that an address is not needed to obtain an inter-domain route to a gateway of a visible superdomain.)

Router Failure Model: A router can undergo failures and recoveries at anytime. We assume failures are fail-stop (i.e. a failed router does not send erroneous messages). When a router x recovers, the variables $WView_x$ and $PendingReq_x$ are lost for all destinations. The cost of each edge in $View_x$ is set to ∞. It becomes up-to-date as the router receives new information from other

[7] This involves time-outs, retransmissions, etc. It requires a transport protocol support such as TCP.

routers.

6 View-Update Protocol

A gateway g, for each ancestor superdomain U, informs other routers of topology changes (i.e. failures, repairs and cost changes) affecting $U{:}g$'s edges. The communication is done by flooding messages. The flooding is restricted to the routers in the parent superdomain of U, since U is visible only to these routers.

Due to the nature of flooding, a router can receive information out of order from a gateway. In order to avoid old information replacing new information, each gateway includes increasing time stamps in the messages it sends. Routers maintain for each gateway the highest received time stamp (in the *timestamp* field in $View_x$), and discard messages with smaller timestamps. Time stamps do not have to be real-time clock values.

Due to superdomain partitioning, messages sent by a gateway may not reach all routers within the parent superdomain, resulting in some routers having out-of-date information. This out-of-date information can cause inconsistencies when the partition is repaired. To eliminate inconsistencies, when a link recovers, the two routers at the ends of the link exchange their views and flood any new information. As usual, information about a superdomain U is flooded over U's parent superdomain.

The view-update protocol uses messages of the following form:

- (Update, *sdid*, *gid*, *timestamp*, *edge-set*)

  Sent by the gateway *gid* to inform other routers about current attributes of edges of *sdid:gid*. *timestamp* indicates the time stamp of *gid*. *edge-set* contains a cost for each edge.

The state maintained by a router x is listed in Figure 16. Note that AdjLocalRouters$_x$ or AdjForeignGateways$_x$ can be empty. $IntraDomainRT_x$ contains a route (next-hop or source)[8] for every reachable node of the domain. We assume that consecutive reads of $Clock_x$ returns increasing values.

Routers also receive and flood messages containing edges of sd-gateways of their ancestor superdomains. This information is used by the query protocol (see Section 5). Also the highest timestamp received from a gateway g of an ancestor superdomain is needed to avoid exchanging

[8] $IntraDomainRT_x$ is a view in case of a link-state routing protocol or a distance table in case of a distance-vector routing protocol.

the messages of g infinitely during flooding.

The events of router x are specified in Figure 16. We use $\texttt{Ancestor}_i(U)$ to denote the superdomain-id of the ith ancestor of U, where $\texttt{Ancestor}_0(U) = U$. In the view-update protocol, a node u uses send operations of the form "Send(m) to v", where m is the message being sent and v is the destination-id. Here, nodes u and v are neighbors, and the message is sent over the physical link $\langle u, v\rangle$. If the link is down, we assume that the packet is dropped.

7 Evaluation

In the superdomain hierarchy (without the query protocol), the number of superdomains in a view is logarithmic in the number of superdomains in the internetwork [10].[9] However, the storage required for a view is proportional not to the number of superdomains in it but to the number of sd-gateways in it. As we have seen, there can be more than one sd-gateway for a superdomain in a view.

In fact, the superdomain hierarchy does not scale-up for arbitrary internetworks; that is, the number of sd-gateways in a view can be proportional to the number of domains in the internetwork. For example, if each domain in a superdomain U has a distinct gateway with a link to outside U, the number of sd-gateways of U would be linear in the number of domains in U.

The good news is that the superdomain hierarchy does scale-up for realistic internetwork topologies. A sufficient condition for scaling is that each superdomain has at most $\log N_D$ sd-gateways; this condition is satisfied by realistic internetworks since most domain interconnections are "hierarchical connections" i.e. between backbones and regionals, between regionals and MANs, and so on.

In this section, we present an evaluation of the scaling properties of the superdomain hierarchy and the query protocol. To evaluate any inter-domain routing protocol, we need a model in which we can define internetwork topologies, policy/ToS constraints, inter-domain routing hierarchies, and evaluation measures (e.g. memory and time requirements). We have recently developed such a model [3]. We first describe our model, and then use it to evaluate our superdomain hierarchy. Our evaluation measures are the amount of memory required at the routers, and the amount of

[9] Even though the results in [10] were for intra-domain routing, it is easy to show that the analysis there holds for inter-domain routing as well.

time needed to construct a path.

7.1 Evaluation Model

We first describe our method of generating topologies and policy/ToS constraints. We then describe the evaluation measures.

Generating Internetwork Topologies

For our purposes, an internetwork *topology* is a directed graph where the nodes correspond to domains and the edges correspond to domain-level connections. However, an arbitrary graph will not do. The topology should have the characteristics of a real internetwork, like the Internet. That is, it should have backbones, regionals, MANS, LANS, etc.; there should be hierarchical connections, but some "non-hierarchical" connections should also be present.

For brevity, we refer to backbones as class 0 domains, regionals as class 1 domains, metropolitan-area domains and providers as class 2 domains, and campus and local-area domains as class 3 domains. A (strictly) hierarchical interconnection of domains means that class 0 domains are connected to each other, and for $i > 0$, class i domains are connected to class $i-1$ domains. As mentioned above, we also want some "non-hierarchical" connections, i.e., domain-level edges between domains irrespective of their classes (e.g. from a campus domain to another campus domain or to a backbone domain).

In reality, domains span geographical regions and domain-level edges are often between domains that are geographically close (e.g. University of Maryland campus domain is connected to SURANET regional domain which are both in the east coast). We also want some edges that are between far domains. A class i domain usually spans a larger geographical region than a class $i+1$ domain. To generate such interconnections, we associate a "region" attribute to each domain. The intention is that two domains with the same region are geographically close.

The *region* of a class i domain has the form $\mathtt{r}_0.\mathtt{r}_1.\cdots.\mathtt{r}_i$, where the r_j's are integers. For example, the region of a class 3 domain can be `1.2.3.4`. For brevity, we refer to the region of a class i domain as a class i region.

Note that regions have their own hierarchy which should not be confused with the superdomain hierarchy. Class 0 regions are the top level regions. We say that a class i region $\mathtt{r}_0.\mathtt{r}_1.\cdots.\mathtt{r}_{i-1}.\mathtt{r}_i$

is *contained* in the class $i-1$ region $x_0.x_1.\cdots.x_{i-1}$ (where $i > 0$). Containment is transitive. Thus region 1.2.3.4 is contained in regions 1.2.3, 1.2 and 1.

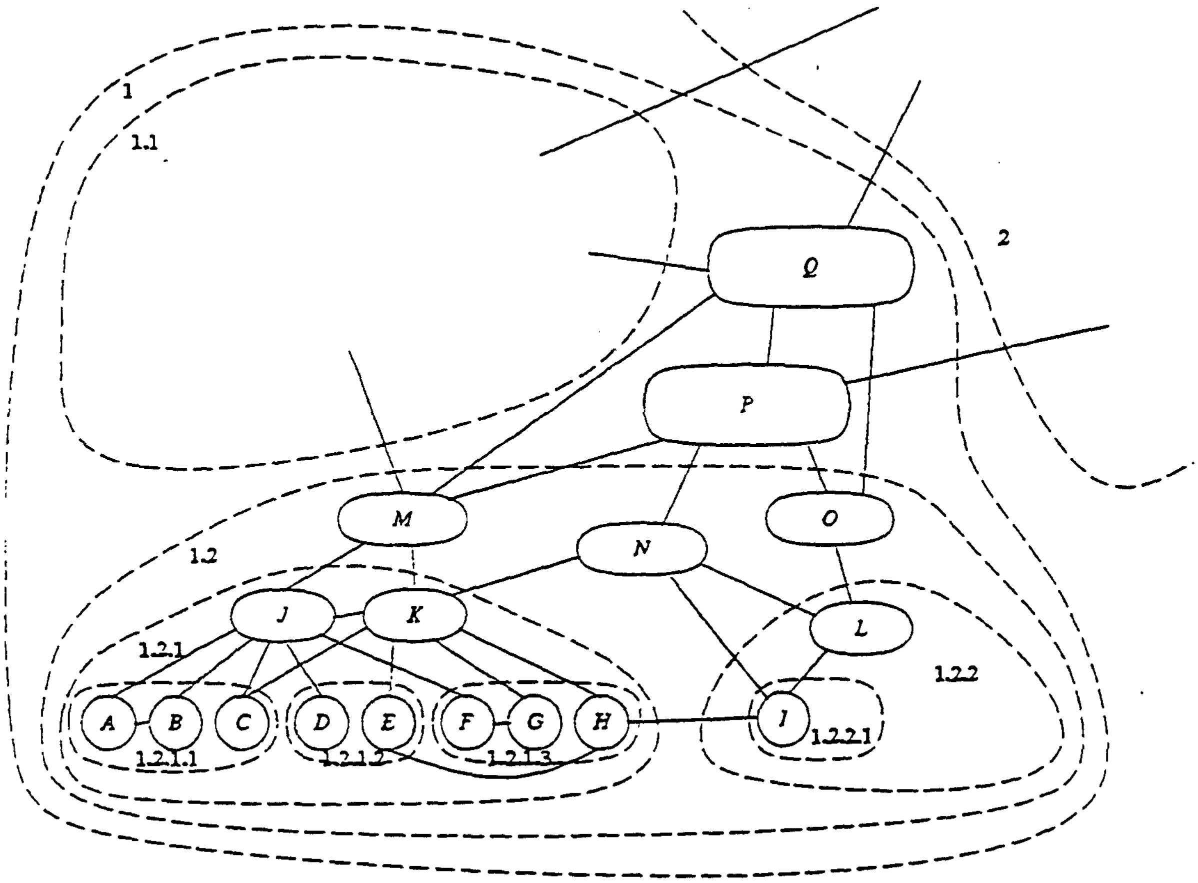


Figure 11: Regions

Given any pair of domains, we classify them as local, remote or far, based on their regions. Let X be a class i domain and Y a class j domain, and without loss of generality let $i \leq j$. X and Y are *local* if they are in the same class i region. For example in Figure 11, A is local to B, C, J, K, M, N, O, P, and Q. X and Y are *remote* if they are not in the same class i region but they are in the same class $i-1$ region, or if $i = 0$. For example in Figure 11, some of the domains A is remote to are D, E, F, and L. X and Y are *far* if they are not local or remote. For example in Figure 11, A is far to I.

We refer to a domain-level edge as *local* (*remote*, or *far*) if the two domains it connects are local (remote, or far).

We use the following procedure to generate internetwork topologies:

- We first specify the number of domain classes, and the number of domains in each class.
- We next specify the regions. Note that the number of region classes equals the number of domain classes. We specify the number of class 0 regions. For each class $i > 0$, we specify a *branching factor*, which creates that many class i regions in each class $i - 1$ region. (That is, if there are two class 0 regions and the class 1 branching factor equals three, then there are six class 1 regions.)
- For each class i, we randomly map the class i domains into the class i regions. Note that several domains can be mapped to the same region, and some regions may have no domain mapped into them.
- For every class i and every class j, $j \geq i$, we specify the number of local, remote and far edges to be introduced between class i domains and class j domains. The end points of the edges are chosen randomly (within the specified constraints).
- We ensure that the internetwork topology is connected by ensuring that the subgraph of class 0 domains is connected, and each class i domain, for $i > 0$, is connected to a local class $i - 1$ domain.
- Each domain has one gateway. So all neighbors of a domain are connected via this gateway. This is for simplicity.

Choosing Policy/ToS Constraints

We chose a simple scheme to model policy/ToS constraints. Each domain is assigned a color: *green* or *red*. For each domain class, we specify the percentage of green domains in that class, and then randomly choose a color for each domain in that class.

A *valid route* from a source to a destination is one that does not visit any red intermediate domains; the source and destination domains are allowed to be red.

This simple scheme can model many realistic policy/ToS constraints, such as security constraints and bandwidth requirements. It cannot model some important kinds of constraints, such as delay bounds.

Computing Evaluation Measures

The evaluation measures of most interest for an inter-domain routing protocol are its memory, time and communication requirements. We postpone the precise definitions of the evaluation measures to the next subsection.

The only analysis method we have at present is to numerically compute the evaluation measures for a variety of source-destination pairs. Because we use internetwork topologies of large sizes, it is not feasible to compute for all possible source-destination pairs. We randomly choose a set of source-destination pairs that satisfy the following conditions: (1) the source and destination domains are different stub domains, and (2) there exists a valid path from the source domain to the destination domain in the internetwork topology. (Note that the straight-forward scheme would always find such a path.)

7.2 Application to Superdomain Query Protocol

We use the above model to evaluate our superdomain query protocol for several different superdomain hierarchies. For each hierarchy, we define a set of superdomain-ids and a parent-child relationship on them.

The first superdomain hierarchy scheme is referred to as *child-domains*. Each domain d (regardless of its class) is a level-1 superdomain, also identified as d. In addition, for each backbone d, we create a distinct level-4 superdomain referred to as d-4. For each regional d, we create a distinct level-3 superdomain d-3 and make it a child of a randomly chosen level-4 superdomain e-4 such that d and e are local and connected. For each MAN d, we create a distinct level-2 superdomain d-2 and make it a child of a randomly chosen level-3 superdomain e-3 such that d and e are local and connected. Please see Figure 12.

We next describe how the level-1 superdomains (i.e. the domains) are placed in the hierarchy. A backbone d is placed in, i.e. as a child of, d-4. A regional d is placed in d-3. A MAN d is placed in d-2. A stub d is placed in e-2 such that d and e are local and connected. Please see Figure 12.

The second superdomain hierarchy scheme is referred to as *sibling-domains*. It is identical to *child-domains* except for the placement of level-1 superdomains corresponding to backbones, regionals and MANs. In *sibling-domains*, a backbone d is placed as a sibling of d-4. A regional d is placed as a sibling of d-3. A MAN d is placed as a sibling of d-2. Please see Figure 13.

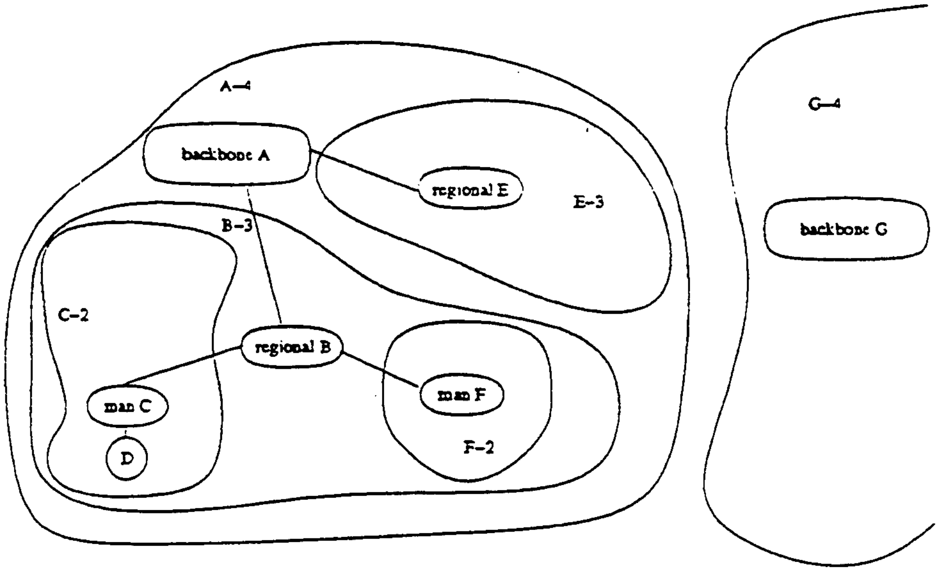


Figure 12: *child-domains*

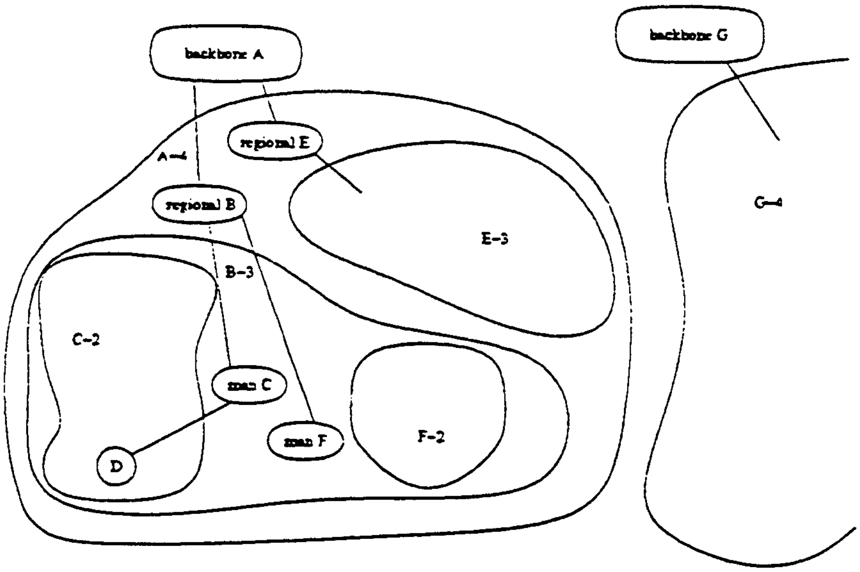


Figure 13: *sibling-domains*

The third superdomain hierarchy scheme is referred to as *leaf-domains*. It is identical to *child-domains* except for the placement of level-1 superdomains corresponding to backbones and regionals.

In *leaf-domains*, backbones and regionals are placed in some level-2 superdomain, as follows. A regional d, if superdomain d-3 has a child superdomain e-2, is placed in e-2. Otherwise, a new level-2 superdomain d-2 is created and placed in d-3. d is placed in d-2. A backbone d, if superdomain d-4 has a child superdomain f-3, is placed in the level-2 superdomain containing the regional f. Otherwise, a new level-3 superdomain d-3 is created and placed in d-4, a new level-2 superdomain d-2 is created and placed in d-3. d is placed in d-2. Please see Figure 14.

Note that in *leaf-domains*, all level-1 superdomains are placed under level-2 superdomains. Whereas other schemes allow some level-1 superdomains to be placed under higher level superdomains.

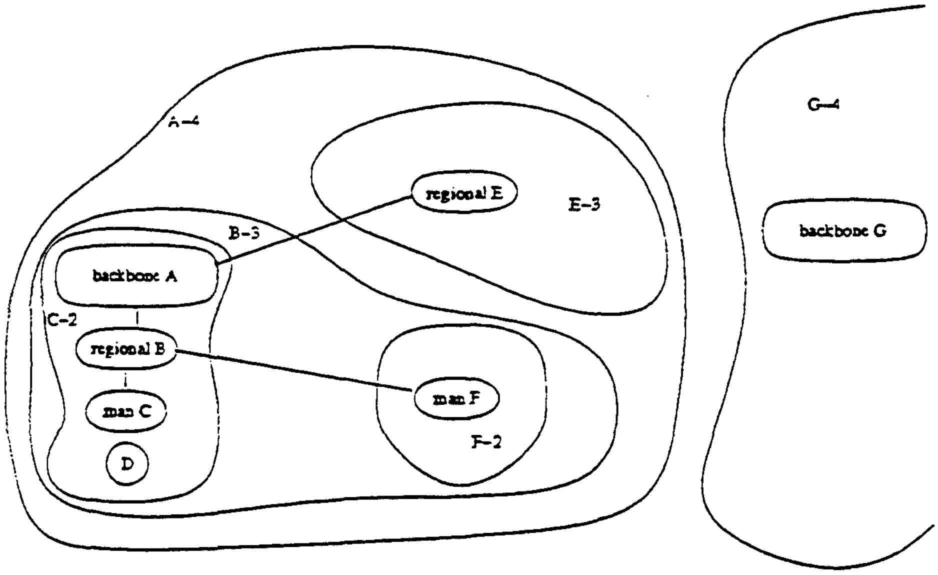


Figure 14: *leaf-domains*

The fourth superdomain hierarchy scheme is referred to as *regions*. In this scheme, the superdomain hierarchy corresponds exactly to the region hierarchy used to generate the internetwork topology. That is, for a class 1 region x there is a distinct level 5 (top level) superdomain x-5. For a class 2 region x.y there is a distinct level 4 superdomain $x.y$-4 placed under level 5 superdomain x-5, and so on. Each domain is placed under the superdomain of its region. Please see Figure 11.

Results for Internetwork 1

The parameters of the first internetwork topology, referred to as Internetwork 1, are shown in Table 1.

Class i	No. of Domains	No. of Regions[10]	% of Green Domains	Edges between Classes i and j			
				Class j	Local	Remote	Far
0	10	4	0.80	0	8	6	0
1	100	16	0.75	0	190	20	0
				1	26	5	0
2	1000	64	0.70	0	100	0	0
				1	1060	40	0
				2	200	40	0
3	10000	256	0.20	0	100	0	0
				1	100	0	0
				2	10100	50	0
				3	50	50	50

Table 1: Parameters of Internetwork 1.

Our evaluation measures were computed for a (randomly chosen but fixed) set of 100,000 source-destination pairs. For a source-destination pair, we refer to the length of the shortest valid path in the internetwork topology as the *shortest-path length*, or *spl* in short. The minimum *spl* of these pairs was 2, the maximum *spl* was 15, and the average *spl* was 6.84.

For each source-destination pair, the set of candidate paths is examined in shortest-first order until either a valid path was found or the set was exhausted and no valid paths were found. For each candidate path, `RequestIView` messages are sent to all candidate superdomains on this path in parallel. All `ReplyIView` messages are received in time proportional to the round-trip time to the farthest of these superdomains. Hence, total time requirement is proportional to the number of candidate paths queried multiplied by the round-trip time to the farthest superdomain in these paths. Let *msgsize* denote the sum of average `RequestIView` message size and average

[10] Branching factor is 4 for all region classes.

Scheme	No query needed	Candidate Paths	Candidate Superdomains
child-domains	220	3.31/13	7.35/38
sibling-domains	220	3/10	6.17/22
leaf-domains	219	6.31/24	15.94/66
regions	544	3.70/12	7.79/30

Table 2: Queries for Internetwork 1.

ReplyIView message size. The number of candidate superdomains queried times *msgsize* indicates the communication capacity required to ship the RequestIView and ReplyIView messages.

Table 2 lists for each superdomain scheme the average and maximum number of candidate paths and candidate superdomains queried. As apparent from the table, *sibling-domains* is superior to other schemes and *leaf-domains* is much worse than the rest. This is because in *leaf-domains*, even if only one domain d in a superdomain U is actually going to be crossed, all descendants of U containing d may need to be queried to obtain a valid path (e.g. to cross backbone A in Figure 14, it may be necessary to query for superdomain A-4, then B-3, then C-2).

	Initial view size		Merged view size	
Scheme	in sd-gateways	in superdomains	in sd-gateways	in superdomains
child-domains	964/1006	42/60	1089/1282	100/298
sibling-domains	1167/1269	70/99	1470/2190	148/337
leaf-domains	963/1006	40/60	1108/1322	130/411
regions	492/715	85/163	1042/2687	158/369

Table 3: View sizes for Internetwork 1.

Table 3 lists for each superdomain scheme the average and maximum of the initial view size and of the merged view size. The initial view size indicates the memory requirement at a router without using the query protocol (i.e. assuming the initial view has a valid path). The merged view size indicates the memory requirement at a router during the query protocol (after finding a valid path). The memory requirement at a router is O(view size in number of sd-gateways × E_G) where E_G is the average number of edges of an sd-gateway. Note that the source does not need to store information about red and non-transit domains in the merged views (other than the ones already in the initial view). The numbers for the merged view sizes in Table 3 take advantage of this.

As apparent from the table, *leaf-domains*, *child-domains* and *regions* scale better than *sibling-domains*. There are two reasons for this. First, placing a backbone (regional or MAN) domain d as a sibling to d-4 (d-3 or d-2) doubles the number of level 4 (3 or 2) superdomains in the views of routers. Second, since these domains have many edges to the domains in their associated superdomains, the end points of each of these edges become sd-gateways of the associated superdomains. Note that *regions* scales much superior to the other schemes in the initial view size. This is because most edges are local (i.e. contained within regions), thus contained completely in superdomains. Hence, their end points are not sd-gateways.

Overall, the *child-domains* and *regions* schemes scale best in space, time and communication requirements. We have repeated the above evaluations for two other internetworks and obtained similar conclusions. The results are in Appendix A.

8 Related Work

In this section, we survey recently proposed inter-domain routing protocols that support ToS and policy routing for large internetworks.

Nimrod [6] and IDPR [16] use the link-state approach with domain-level source routing to enforce policy and ToS constraints and superdomains to solve scaling problem. Nimrod is still in a design stage. Both protocols suffer from loss of policy and ToS information as mentioned in the introduction. A query protocol for Nimrod is being developed to obtain more detailed policy, ToS and topology information.

BGP [12] and IDRP [14] are based on a *path-vector* approach [15]. Here, for each destination domain a router maintains a set of paths, one through each of its neighbor routers. ToS and policy information is attached to these paths. Each router requires $O(N_D \times N_D \times E_R)$ space, where N_D is the average number of neighbor domains for a domain and N_R is the number of routers in the internetwork. For each destination, a router exchanges its best valid path with its neighbor routers. However, a path-vector algorithm may not find a valid path from a source to the destination even if such a route exists [16][11] (i.e. detailed ToS and policy information may be lost). By exchanging k paths to each destination, the probability of detecting a valid path for each source can be increased. But to guarantee detection, either all possible paths should be exchanged (exponential number of paths in the worst case) or source policies should be made public and routers should take this into account when exchanging routes. However, this fix increases space and communication requirements drastically.

IDRP [14] uses superdomains to solve the scaling problem. It exchanges all paths between neighbor routers subject to the following constraint: a router does not inform a neighbor router of a route if usage of the route by the neighbor would violate some superdomain's constraint on the route. IDRP also suffers from loss of ToS and policy information. To overcome this problem, it uses overlapping superdomains: that is, a domain and superdomain can be in more than one parent superdomain. If a valid path over a domain can not be discovered because the constraints of a parent superdomain are violated, the same path may be discovered through another parent superdomain whose constraints are not violated. However, handling ToS and policy constraints in general requires more and more combinations of overlapping superdomains, resulting in more storage requirement.

Reference [9] combines the benefits of path-vector approach and link-state approach by having two modes: An NR mode, which is an extension of IDRP and is used for the most common ToS and policy constraints; and a SDR mode, which is like IDPR and is used for less frequent ToS and policy requests. This study does not address the scalability of the SDR mode. Ongoing work by this group considers a new SDR mode which is not based on IDPR.

Reference [19] suggests the use of multiple addresses for each node, one for each ToS and Policy. This scheme does not scale up. In fact, it increases the storage requirement, since a router maintains a route for each destination address, and there are more addresses with this scheme.

The landmark hierarchy [18, 17] is another approach for solving scaling problem. Here, each router is a landmark with a radius, and routers which are at most radius away from the landmark maintain a route for it. Landmarks are organized hierarchically, such that radius of a landmark increases with its level, and the radii of top level landmarks include all routers. Addressing and

[11] For example, suppose a router u has two paths $P1$ and $P2$ to the destination. Let u have a router neighbor v, which is in another domain. u chooses and informs v of one of the paths, say $P1$. But $P1$ may violate source policies of v's domain, and $P2$ may be a valid path for v.

packet forwarding schemes are introduced. Link-state algorithms can not be used with the landmark hierarchy, and a thorough study of enforcing ToS and policy constraints with this hierarchy has not been done.

In [1], we provided an alternative solution to loss of policy and ToS information that is perhaps more faithful to the original superdomain hierarchy. To handle superdomain-level source routing and topology changes, we augmented each superdomain-level edge (U, V) with the address of an "exit" domain u in U and an "entry" domain v in V. To obtain internal views, we added for each visible superdomain U the edges from U to domains outside the parent of U. Surprisingly, this approach and the gateway-level view approach have the same memory and communication requirements. However, the first approach results in much more complicated protocols.

Reference [2] presents interdomain routing protocols based on a new kind of hierarchy, referred to as the viewserver hierarchy. This approach also scales well to large internetworks and does not lose detail ToS and policy information. Here, special routers called viewservers maintain the view of domains in a surrounding precinct. Viewservers are organized hierarchically such that for each viewserver, there is a domain of a lower level viewserver in its view, and views of top level viewservers include domains of other top level viewservers. Appropriate addressing and route discovery schemes are introduced.

9 Conclusion

We presented a hierarchical inter-domain routing protocol which satisfies policy and ToS constraints, adapts to dynamic topology changes including failures that partition domains, and scales well to large number of domains.

Our protocol achieves scaling in space requirement by using superdomains. Our protocol maintains superdomain-level views with sd-gateways and handles topology changes by using a link-state view update protocol. It achieves scaling in communication requirement by flooding topology changes affecting a superdomain U over U's parent superdomain.

Our protocol does not lose detail in ToS, policy and topology information. It stores both a strong set of constraints and a weak set of constraints for each visible superdomain. If the weak constraints but not the strong constraints of a superdomain U are satisfied (i.e. the aggregation has resulted in loss of detail in ToS and policy information), then some paths through U may be valid.

Our protocol uses a query protocol to obtain a more detailed "internal" view of such superdomains, and searches again for a valid path. Our evaluation results indicate that the query protocol can be performed using 15% extra space.

One drawback of our protocols is that to obtain a source route, views are merged at or prior to the connection setup, thereby increasing the setup time. This drawback is not unique to our scheme [7, 16, 6, 9]. There are several ways to reduce this setup overhead. First, source routes to frequently used destinations can be cached. Second, the internal views of frequently queried superdomains can be cached at routers close to the source domain. Third, better heuristics to choose candidate paths and candidate superdomains to query can be developed.

We also described an evaluation model for inter-domain routing protocols. This model can be applied to other inter-domain routing protocols. We have not done so because precise definitions of the hierarchies in these protocols are not available. For example, to do a fair evaluation of IDPR[16], we need precise guidelines for how to group domains into superdomains, and how to choose between the strong and weak methods when defining policy/ToS constraints of superdomains. In fact, these protocols have not been evaluated in a way that we can compare them to the superdomain hierarchy.

References

[1] C. Alaettinoğlu and A. U. Shankar. Hierarchical Inter-Domain Routing Protocol with On-Demand ToS and Poicy Resolution. In Proc. *IEEE International Conference on Networking Protocols '93*, San Fransisco, California, October 1993.

[2] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A New Inter-Domain Routing Protocol and its Evaluation. Technical Report UMIACS-TR-93-98, CS-TR-3151, Department of Computer Science, University of Maryland, College Park, October 1993. Earlier version CS-TR-3033, February 1993.

[3] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A New Inter-Domain Routing Protocol. In Proc. *IEEE INFOCOM '94*, Toronto, Canada, June 1994. To appear.

[4] A. Bar-Noy and M. Gopal. Topology Distribution Cost vs. Efficient Routing in Large Networks. In Proc. *ACM SIGCOMM '90*, pages 242–252, Philadelphia, Pennsylvania, September 1990.

[5] L. Breslau and D. Estrin. Design of Inter-Administrative Domain Routing Protocols. In Proc. *ACM SIGCOMM '90*, pages 231–241, Philadelphia, Pennsylvania, September 1990.

[6] I. Castineyra, J. N. Chiappa, C. Lynn, R. Ramanathan, and M. Steenstrup. The Nimrod Routing Architecture. Internet Draft., March 1994. Available by anonymous ftp from `research.ftp.com:pub/nimrod`.

[7] D.D. Clark. Policy routing in Internet protocols. Request for Comment RFC-1102, Network Information Center, May 1989.

[8] D. Estrin. Policy requirements for inter Administrative Domain routing. Request for Comment RFC-1125, Network Information Center, November 1989.

[9] D. Estrin, Y. Rekhter, and S. Hotz. Scalable Inter-Domain Routing Architecture. In Proc. *ACM SIGCOMM '92*, pages 40–52, Baltimore, Maryland, August 1992.

[10] L. Kleinrock and F. Kamoun. Hierarchical Routing for Large Networks. *Computer Networks and ISDN Systems*, (1):155–174, 1977.

[11] B.M. Leiner. Policy issues in interconnecting networks. Request for Comment RFC-1124, Network Information Center, September 1989.

[12] K. Lougheed and Y. Rekhter. Border Gateway Protocol (BGP). Request for Comment RFC-1105, Network Information Center, June 1989.

[13] R. Perlman. Hierarchical Networks and Subnetwork Partition Problem. *Computer Networks and ISDN Systems*, 9:297–303, 1985.

[14] Y. Rekhter. Inter-Domain Routing Protocol (IDRP). Available from the author., 1992. T.J. Watson Research Center, IBM Corp.

[15] K. G. Shin and M. Chen. Performance Analysis of Distributed Routing Strategies Free of Ping-Pong-Type Looping. *IEEE Transactions on Computers*, 1987.

[16] M. Steenstrup. An Architecture for Inter-Domain Policy Routing. Request for Comment RFC-1478, Network Information Center, July 1993.

[17] P. F. Tsuchiya. The Landmark Hierarchy: Description and Analysis, The Landmark Routing: Architecture Algorithms and Issues. Technical Report MTR-87W00152, MTR-87W00174, The MITRE Corporation, McLean, Virginia, 1987.

[18] P. F. Tsuchiya. The Landmark Hierarchy:A New Hierarchy For Routing In Very Large Networks. In Proc. *ACM SIGCOMM '88*, August 1988.

[19] P. F. Tsuchiya. Efficient and Robust Policy Routing Using Multiple Hierarchical Addresses. In Proc. *ACM SIGCOMM '91*, pages 53–65, Zurich, Switzerland, September 1991.

A Results for Other Internetworks

Results for Internetwork 2

The parameters of the second internetwork topology, referred to as Internetwork 2, are the same as the parameters of Internetwork 1 but a different seed is used for the random number generation.

Our evaluation measures were computed for a set of 100,000 source-destination pairs. The minimum *spl* of these pairs was 1, the maximum *spl* was 14, and the average *spl* was 7.13.

Table 5 and Table 4 shows the results. Similar conclusions as in the case of Internetwork 1 hold.

Results for Internetwork 3

The parameters of the third internetwork topology, referred to as Internetwork 3, are shown in Table 6. Internetwork 3 is more connected, more class 0, 1 and 2 domains are green, and more class 3 domains are red. Hence, we expect bigger view sizes in number of sd-gateways.

Scheme	No query needed	Candidate Paths	Candidate Superdomains
child-domains	205	4.52/20	10.22/47
sibling-domains	205	3.01/8	6.50/21
leaf-domains	205	8.80/32	21.34/82
regions	640	3.52/10	7.85/28

Table 4: Queries for Internetwork 2.

	Initial view size		Merged view size	
Scheme	in sd-gateways	in superdomains	in sd-gateways	in superdomains
child-domains	958/1012	43/60	1079/1269	118/306
sibling-domains	1153/1283	72/101	1480/2169	160/324
leaf-domains	956/1009	41/58	1095/1281	156/387
regions	624/1024	110/231	1356/3578	206/435

Table 5: View sizes for Internetwork 2.

Our evaluation measures were computed for a set of 100,000 source-destination pairs. The minimum *spl* of these pairs was 1, the maximum *spl* was 11, and the average *spl* was 5.95.

Table 8 and Table 7 shows the results. Similar conclusions as in the cases of Internetwork 1 and 2 hold.

[12] Branching factor is 4 for all domain classes.

Class i	No. of Domains	No. of Regions[12]	% of Green Domains	Edges between Classes i and j			
				Class j	Local	Remote	Far
0	10	4	0.85	0	8	7	0
1	100	16	0.80	0	190	20	0
				1	50	20	0
2	1000	64	0.75	0	500	50	0
				1	1200	100	0
				2	200	40	0
3	10000	256	0.10	0	300	50	0
				1	250	100	0
				2	10250	150	50
				3	200	150	100

Table 6: Parameters of Internetwork 3.

Scheme	No query needed	Candidate Paths	Candidate Superdomains
child-domains	142	3.99/29	7.70/43
sibling-domains	142	2.95/10	5.39/22
leaf-domains	142	9.65/70	18.99/103
regions	676	3.47/17	6.25/21

Table 7: Queries for Internetwork 3.

	Initial view size		Merged view size	
Scheme	in sd-gateways	in superdomains	in sd-gateways	in superdomains
child-domains	2160/2239	43/60	2354/2647	107/348
sibling-domains	2365/2504	72/101	2606/3314	148/356
leaf-domains	2159/2236	41/58	2386/2645	160/648
regions	1107/1644	110/231	1850/3559	194/436

Table 8: View sizes for Internetwork 3.

```
Variables:
  $View_x$. Dynamic view of $x$.
  $WView_x(d\_address)$. Temporary view of $x$. $d\_address$ is the destination address.
      Used for merging internal views of superdomains to the view of $x$.
  $PendingReq_x(d\_address)$. Integer. $d\_address$ is the destination address.
      Number of outstanding request messages.
Events:
  $Request_x(d\_address)$        {Executed when $x$ wants a valid domain-level source route}
     allocate $WView_x(d\_address) := View_x$; allocate $PendingReq_x(d\_address) := 0$;
     $search_x(d\_address)$;

where
  $search_x(d\_address)$
     if there is a valid path to $d\_address$ in $WView_x(d\_address)$ then
        $result$ := shortest valid path;
        deallocate $WView_x(d\_address)$, $PendingReq_x(d\_address)$;
        return $result$;
     else if there is a candidate path to $d\_address$ in $WView_x(d\_address)$ then
           Let $cpath = \langle U_0{:}g_{0_0}, \ldots, U_0{:}g_{0_{n_0}}, U_1{:}g_{1_0}, \ldots, U_1{:}g_{1_{n_1}}, \cdots, U_m{:}g_{m_0}, \ldots, U_m{:}g_{m_{n_m}} \rangle$
                be the shortest candidate path;
           for $U_i$ in $cpath$ such that $U_i$ is candidate  do
               ReliableSend(RequestIView, $U_i, g_{i_0}, address(x), d\_address$) to $g_{i_0}$
               $PendingReq_x(d\_address) := PendingReq_x(d\_address) + 1$;
        else
           deallocate $WView_x(d\_address)$, $PendingReq_x(d\_address)$;
           return failure;
        endif
     endif

  $TimeOut_x(d\_address)$        {Executed after a time-out period and $PendingReq_x(d\_address) \neq 0$.}
     deallocate $WView_x(d\_address)$, $PendingReq_x(d\_address)$;
     return failure;
```

Figure 15: view-query protocol: State and events of a router x. (Figure continued on next page.)

```
$Receive_x$(RequestIView, $sdid, x, s\_address, d\_address$)
    ReliableSend(ReplyIView, $sdid, x, IView_x(U), d\_address$) to $s\_address$;

$Receive_x$(ReplyIView, $sdid, gid, iview, d\_address$)
    if $PendingReq_x(d\_address) \neq 0$ then          {No time-out happened}
        $PendingReq_x(d\_address) := PendingReq_x(d\_address) - 1$;
                {merge internal view}
        delete $\langle sdid, *, *, * \rangle$ from $WView_x$;
        for $\langle child,\ scons,\ wcons,\ gateway\text{-}set \rangle$ in $iview$ do
            if $\neg\exists\langle child, *, *, * \rangle \in WView_x$ then
                insert $\langle child,\ scons,\ wcons,\ gateway\text{-}set \rangle$ in $WView_x$;
            else
                for $\langle gid,\ ts,\ edge\text{-}set \rangle$ in $gateway\text{-}set$ do
                    if $\exists\langle gid,\ timestamp,\ * \rangle \in Gateways\&Edges_x(child) \wedge ts > timestamp$ then
                        delete $\langle gid,\ *,\ * \rangle$ from $Gateways\&Edges_x(child)$;
                    endif;
                    if $\neg\exists\langle gid,\ *,\ * \rangle \in Gateways\&Edges_x(child)$ then
                        insert $\langle gid,\ ts,\ edge\text{-}set \rangle$ to $Gateways\&Edges_x(child)$;
                    endif
            endif
        if $PendingReq_x(d\_address) = 0$ then          {All pending replies are received}
            $search_x(d\_address)$;
        endif
    endif
```

Figure 15: view-query protocol: State and events of a router x. (cont.)

Constants:

AdjLocalRouters$_x$. ($\subseteq$ NodeIds). Set of neighbor routers in x's domain.

AdjForeignGateways$_x$. ($\subseteq$ NodeIds). Set of neighbor routers in other domains.

Ancestor$_i(x)$. ($\subseteq$ SuperDomainIds). ith ancestor of x.

Variables:

$View_x$. Dynamic view of x.

$IntraDomainRT_x$. Intra-domain routing table of x. Initially contains no entries.

$Clock_x$: Integer. Clock of x.

Events:

```
Receive$_x$(Update, $sdid$, $gid$, $ts$, $edge\text{-}set$) from $sender$
    if $\exists(gid, timestamp, *) \in Gateways\&Edges_x(sdid) \wedge ts > timestamp$ then
        delete $\langle gid, *, * \rangle$ from $Gateways\&Edges_x(sdid)$;
    endif;
    if $\neg\exists\langle gid, *, * \rangle \in Gateways\&Edges_x(sdid)$ then
        $flood_x$((Update, $sdid$, $gid$, $ts$, $edge\text{-}set$));
        insert $\langle gid, ts, edge\text{-}set \rangle$ to $Gateways\&Edges_x(sdid)$;
        $update\_parent\_domains_x$(level($sdid$) + 1);
    endif

where
  $update\_parent\_domains_x(startinglevel)$
    for $level := startinglevel$ to number of levels in the hierarchy  do
        $sdid$ := Ancestor$_{level}(x)$;
        if $x \in$ Gateways($sdid$) then
            $edge\text{-}set$ := aggregate edges of $sdid{:}x$ using $View_x$, $IntraDomainRT_x$ and links of $x$;
            $timestamp = Clock_x$;
            $flood_x$((Update, $sdid$, $x$, $timestamp$, $edge\text{-}set$));
            delete $\langle x, *, * \rangle$ from $Gateways\&Edges_x(sdid)$;
            insert $\langle x, timestamp, edge\text{-}set \rangle$ to $Gateways\&Edges_x(sdid)$;
        endif

  $Do\_Update_x$          {Executed periodically and upon a change in $IntraDomainRT_x$ or links of $x$}
    $update\_parent\_domains_x(1)$

  $Link\_Recovery_x(y)$          {$\langle x,y \rangle$ is a link. Executed when $\langle x,y \rangle$ recovers.}
    for all $\langle sdid, *, *, * \rangle$ in $View_x$ do
        if $\exists i$ : Ancestor$_i(y)$ = Ancestor$_1(sdid)$ then
            for all $\langle gid, timestamp, edge\text{-}set \rangle$ in $Gateways\&Edges_x(sdid)$ do
                Send((Update, $sdid$, $gid$, $timestamp$, $edge\text{-}set$)) to $y$;
        endif

  $flood_x(packet)$
    for all $y \in$ AdjLocalRouters$_x$ do
        Send($packet$) to $y$;
    for all $y \in$ AdjForeignGateways$_x$ $\wedge\ \exists i$ : Ancestor$_i(y)$ = Ancestor$_1(packet.sdid)$ do
        Send($packet$) to $y$;
```

Figure 16: view-update protocol: State and events of a router x.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
June 20, 1994

3. REPORT TYPE AND DATES COVERED
Technical

4. TITLE AND SUBTITLE
Hierarchical Inter-Domain Routing Protocol with
On-Demand ToS and Policy Resolution

5. FUNDING NUMBERS
DASG-60-92-C-0055

6. AUTHOR(S)
Cengiz Alaettinoglu and A. Udaya Shankar

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Maryland
A. V. Williams Building
Department of Computer Science
College Park, MD 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
CS-TR 3299
UMIACS-TR-94-73

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Phillips Labs
3550 Aberdeen Ave. SE
Kirtland AFB, NM 87117-5776

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Traditional inter-domain routing protocols based on superdomains maintain either "strong" or "weak" ToS and policy constraints for each visible superdomain. With strong constraints, a valid path may not be found even though one exists. With weak constraints, an invalid domain-level path may be treated as a valid path.

We present an inter-domain routing protocol based on superdomains, which always finds a valid path if one exists. Both strong and weak constraints are maintained for each visible superdomain. If the strong constraints of the superdomains on a path are satisfied, then the path is valid. If only the weak constraints are satisfied for some superdomains on the path, the source uses a query protocol to obtain a more detailed "internal" view of these superdomains, and searches again for a valid path. Our protocol handles topology changes, including node/link failures that partition superdomains. Evaluation results indicate our protocol scales well to large internetworks.

14. SUBJECT TERMS (Routing Protocols);(Computer Network Routing Protocols)
(Computer-Communication Networks): Network Architecture and
Design- packet networks; store and forward networks; (Computer
Communication Networks:Network Protocols-protocol architectur

15. NUMBER OF PAGES
34

16. PRICE CODE
N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

DEPARTMENT OF THE AIR FORCE
PHILLIPS LABORATORY (AFMC)

30 Jul 97

MEMORANDUM FOR DTIC/OCP

FROM: Phillips Laboratory/CA
3550 Aberdeen Ave SE
Kirtland AFB, NM 87117-5776

SUBJECT: Public Releasable Abstracts

1. The following technical report **abstracts** have been cleared by Public Affairs for unlimited distribution:

PL-TR-96-1126, Pt 1	ADB222369	PL 97-0685 (clearance number)
PL-TR-96-1126, Pt 2	ADB222192	PL 97-0685

2. Any questions should be referred to Jan Mosher at DSN 246-1328.

Janet E. Mosher
JANET E. MOSHER
Writer/Editor

cc:
PL/TL/DTIC (M Putnam)